PETER HEUER

6502

Microcomputer Programmierung

ISBN 3-921682-43-6

1. Auflage 1979

Gedruckt in der Bundesrepublik Deutschland – Printed in West Germany – Imprime' en RFA

6502 AUFBAU UND PROGRAMMIERUNG

eine leicht verständliche Einführung
in die Programmierung von
Microcomputern



Inhaltsverzeichnis

QUELLENVERZEICHNIS

R6500 Microcomputer System Manual von Rockwell
66500 Microcomputer System Programming Manual von Rockwell
Bilder von Fa. Rockwell und Synertec, ITT, Commodore PET und KIM-1 (MOS-Technology).

Anschriften der 65XX Hersteller:

1. MOS-Technology, 950 Rittenhouse Rd. Norristown, PA19401
   Tel.: 001/215/666/9750

In Deutschland:
   MOS-Technology
   Frankfurter Str. 171 – 175
   6078 Neu-Isenburg

2. Rockwell International GmbH, Microelectronik Devices
   Fraunhoferstr. 11
   D-8033 Martinsried

3. Synertec Systems Corporation
   P.O. Box 552
   Santa Clara, CA 95052
   Tel.: 001/408/988/5600

QUELLENVERZEICHNIS

Vorwort

Wenn man sich entschließt, ein Buch zu schreiben, so bedarf das natürlich einer Begründung, die ich dem Leser nicht vorenthalten will.

Die Microcomputer erobern mit großem Tempo den kommerziellen EDV Bereich, den Hobbyelektronikbereich und nicht zuletzt die Schulen und Hochschulen. Die Begründung ist einfach in der negativen Preisentwicklung der Hardware zu sehen. Die Hardwarepreise sind ein Bruchteil der Preise älterer Computer mit gleicher Leistungsfähigkeit.

Nun werden viele Menschen in Beruf, Freizeit oder Ausbildung mit ihnen bisher unbekannten, ganz neuen Problemkreisen konfrontiert. Sie sind nicht in der Lage, ohne entsprechende Unterstützung die neuen Problemstellungen der Softwareentwicklung zu meistern. Sie werden also versuchen, anhand geeigneter Literatur sich auszubilden. Hierbei will dieses Buch behilflich sein.

In der Einführung dieses Buches werden die Zielgruppen, die Ziele und der Aufbau näher erläutert. Lesen Sie bitte erst die Einführung durch, bevor Sie sich anderen Kapiteln zuwenden.

Der Verfasser

Vorwort

[illegible]

[illegible]

[illegible]

[illegible]

Der Verfasser

0. Die Einführung

Wie der Untertitel besagt, will dieses Buch eine leicht verständliche Einführung in die Programmierung von Microcomputern sein. Es ist also nicht ein systematisch aufgebautes und vollständiges Lehrbuch.

Da das Gebiet der Programmierung (Softwareentwicklung) sehr umfangreich und komplex ist, kann es nicht zufriedenstellend und vollständig in einem Buch abgehandelt werden. Man muß sich bewußt auf bestimmte Problemstellungen beschränken. Dieses Buch beschränkt sich auf eine Einleitung in die Programmiertechnik und eine ausführliche Darstellung der Entwicklung von Anwenderprogrammen im Maschinencode und Assemblercode.

Die Stärke dieses Buches liegt in seinen vielen Programmbeispielen. Es sollen nämlich anhand vieler Beispiele die Programmiertechniken eingeübt werden.

Da dieses Buch auch Leser ansprechen will, die wenig mathematische Voraussetzungen haben, wurden bewußt wenig mathematische und viele andersartige Programme (mehrere Spiele) ausgewählt.

Dieses Buch will einen möglichst engen Kontakt zur Praxis herstellen. Aus diesem Grunde wurde auch ein bestimmter Microprozessor, (der 6502) ausgewählt und nicht mehrere vorgestellt. Der Umfang des Buches würde gesprengt, wenn man seinen Charakter nicht verändern wollte.

Dieses Buch will dennoch nicht nur den Besitzer oder Programmierer eines Microcomputers mit dem 6502 als CPU ansprechen. Sein Ziel ist es, gerade an einem konkreten Microprozessor die Programmierung so ausführlich einzuüben, um dem Leser auch die Programmierung eines anderen Microprocessors zu erleichtern.

Grundsätzlich sollte man sich davor hüten, nur die Programmierung eines bestimmten Microprocessors zu erlernen und alle anderen zu vernachlässigen.

Wer soll dieses Buch wie lesen?

Der Leserkreis wird sicherlich sehr inhomogen mit sehr unterschiedlichen Eingangsvoraussetzungen sein. Jeder Leser sollte Grundkenntnisse in der Digitalelektronik besitzen.

Man kann ein Buch nach mehreren Methoden durcharbeiten.

Einmal möchte man nur einen Überblick gewinnen. Dann kann man getrost die einzelnen Beispiele in Kapitel 4. durchlesen, und wenn Fragen auftreten, die Kapitel, in denen die entsprechenden Themenkreise angesprochen werden, sich vornehmen. Um einen Gewinn aus diesem Buch zu ziehen, muß man sich nicht unbedingt einen eigenen Microcomputer anschaffen.

Will man intensiver arbeiten und steckt man sich als Lernziel, eigene Programme entwickeln zu können, so genügt ein Durchlesen nicht. Man sollte dann jedes Beispiel in Kapitel 4. genau durchdenken und versuchen, alle Fragen möglichst zufriedenstellend zu klären. Manchem Leser werden einzelne Abschnitte zu knapp gestaltet sein. Ihn muß man dann auf die entsprechende Spezialliteratur verweisen. Auch für ein intensiveres Durcharbeiten ist nicht unbedingt die Anschaffung eines Microcomputers notwendig, aber doch empfehlenswert. Erst wenn die Beispiele praktisch nachgearbeitet werden, lernen Sie alle Einzelheiten kennen.

Auf Übungsaufgaben wurde verzichtet. Allerdings sollen die Beispiele zu selbständiger Arbeit anregen. Sicherlich werden Sie auch bald mehr Programmierthemen haben als Zeit, diese zu lösen.

1. Die Hardware; einige Microcomputer

Hardware ist ein englischer Fachbegriff für den es keinen entsprechenden deutschen Fachausdruck gibt. Unter Hardware versteht man alles, was real vorhanden ist (was man anfassen kann). Zur Hardware gehören: die elektronischen Bauteile, die Verdrahtung, die Datenstationen, das Gehäuse u. s. w. eines Computers.

Die Hardware von Microcomputern besteht meist nur aus wenigen Platinen mit relativ wenigen, hochintegrierten Bausteinen. Diese recht unscheinbare Hardware hat aber einiges „in sich", was der unvorbereitete Betrachter sicherlich nicht einmal erträumen kann.

In diesem Kapitel können wir natürlich nicht einen erschöpfenden Einblick in die Hardware von Microcomputern gewinnen. Dazu muß auf die entsprechenden Handbücher der verschiedenen Hersteller verwiesen werden. Dieses Kapitel will dem Leser einen Überblick geben, soweit dieser für das Verständnis der nachfolgenden Programme notwendig ist. Andererseits sollen hier einige, für den Privatgebrauch besonders gut geeignete, Microcomputer besprochen werden. Alle Microcomputer besitzen als Microprozessor den 6502.

1.1 Einfache Microcomputer

Die EDV-Anlagen werden grob in die folgenden Gruppen eingeteilt:

Großrechenanlagen,
Rechenanlagen der mittleren Datentechnik (MDT),
Minicomputer,
Microcomputer.

Die einzelnen Gruppen sind nicht scharf gegeneinander abgegrenzt. Prinzipiell unterscheidet sich ein Microcomputer in seiner Funktion nicht von einer Großrechenanlage. Natürlich bestehen zwischen beiden Gruppen gravierende Unterschiede.

Unter Microcomputern versteht man Computer, die aus nur wenigen, stark integrierten Bausteinen bestehen. Die Microcomputer unter-

scheiden sich von den Minicomputern in einer größeren Rechenzeit, bedingt durch die starke Integration und eine kleinere Wortlänge. Zur Zeit gibt es nur wenige 16 bit Microprozessoren. Die meisten Microcomputer haben eine Wortlänge von 8 bit. Die Minicomputer haben eine Wortlänge von 12 bit und mehr.

Der Aufbeu eines Microcomputers mit dem Microprozessor 6502

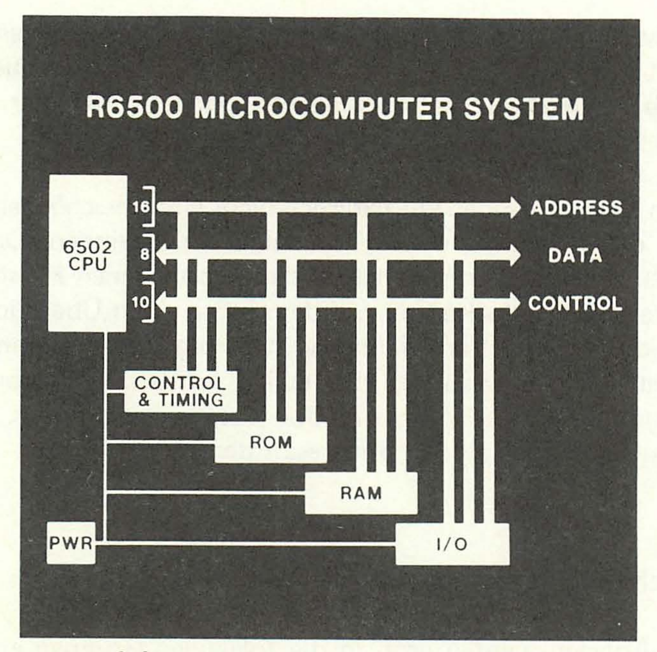


Abb. 1.1-1

Die Abbildung 1.1-1 zeigt das Grundgerüst eines Microcomputers der Familie 6500.

Das Herz des Microcomputers ist die CPU (central processing unit), die die Operationen ausführt und die peripheren Elemente steuert. Bei Microcomputern ist die CPU ein Baustein, der Microprozessor. Der Microprozessor 6502 besitzt einen 16 bit breiten Adressbus, einen 8 bit breiten Datenbus und einen 10 bit breiten Controlbus.

Der Microprozessor allein ist kein funktionsfähiges Element. Er benötigt zusätzlich einige Peripherie. Die Kontroll- und Steuereinheit

(control and timing unit) besteht im wesentlichen aus einem Taktgenerator, der die Operationszyklen der CPU steuert. Als weitere Peripherie benötigt die CPU Speicher. Diese Speicher enthalten Programme, die die Abläufe und Aufgaben des Microcomputers definieren. Programme, die nicht verändert werden, sind in Festwertspeichern (ROM 's) abgelegt. Programme, die veränderbar sein sollen, werden kurzfristig in Schreib-/Lesespeichern (RAM' s) gespeichert. Im RAM-Bereich des Microcomputers befindet sich auch der Datenspeicher. Die meisten Daten werden nur kurzzeitig dort zwischengespeichert. Die Kommunikation mit der Außenwelt findet über die Ein- und Ausgabekanäle statt. Hier werden Terminals, externe Massenspeicher, Meßgeräte, Plotter u. a. m. über entsprechende Anpaßschaltungen (Interface) angeschlossen.

1.1.1 Der KIM, ein typischer Microcomputer

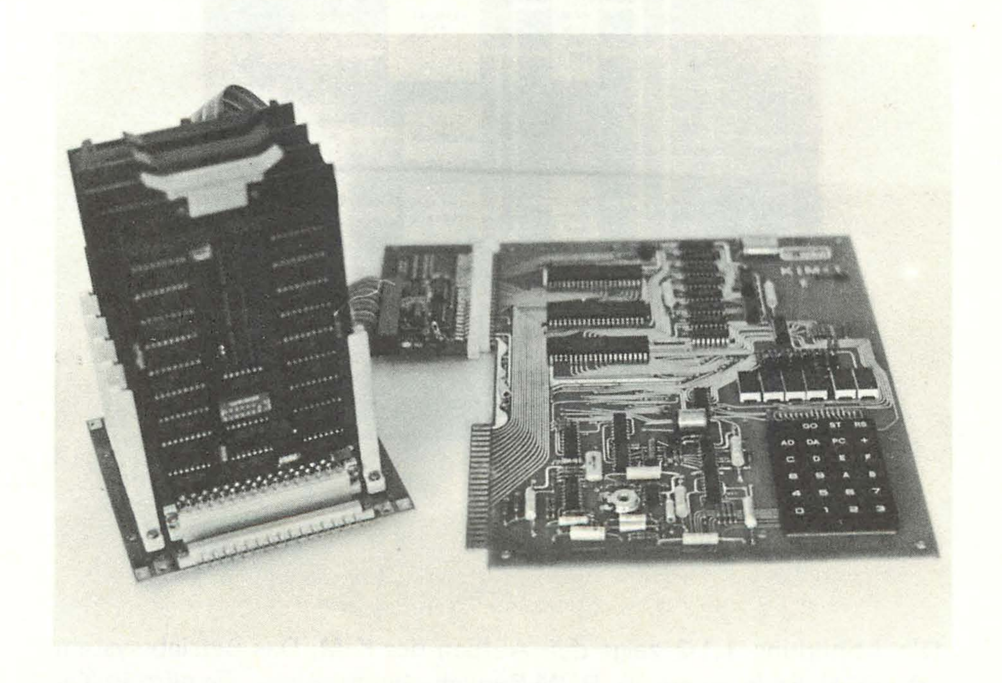


Abbildung: KIM-1

Der KIM ist ein sehr leistungsfähiger Microcomputer, der anfangs nur vom Ersthersteller des 6502, MOS Technologie, angeboten wurde.

Alle notwendigen Bausteine sind auf einer ca. 20 cm x 28 cm großen Leiterplatine untergebracht. Nur die Stromversorgung muß extern zugeschaltet werden. Am Beispiel des KIM ist ersichtlich, wie wenig Hardware zum Aufbau eines sehr leistungsfähigen Microcomputers notwendig ist. Er ist zur Zeit der wohl preisgünstigste Microcomputer dieser Leistungsklasse, weshalb er auch schon große Verbreitung gefunden hat.

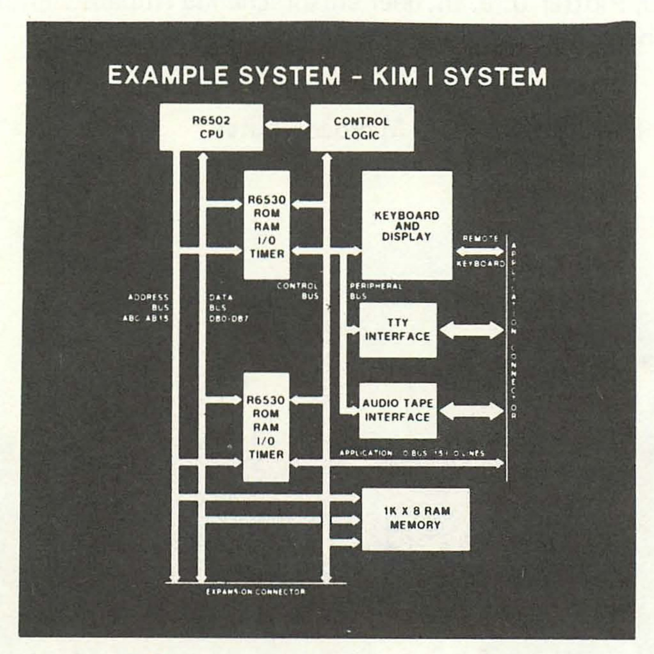


Abb.: 1.1-2

Die Abbildung 1.1-2 zeigt den Aufbau des KIM. Das Betriebssystem (Monitor) ist in einem 2k ROM-Bereich abgespeichert. Es wird in Kapitel 2.2 kurz vorgestellt. Je 1 k ROM (1024 byte) sind in den beiden Bausteinen 6530 enthalten.

Der RAM-Bereich des KIM umfaßt auch 1k. Fast der gesamte RAM-Bereich steht dem Anwender zur Verfügung als Programm- und Datenspeicher.

Der KIM ist ohne externe Datenstation (Terminal) funktionsfähig, da auf der Platine ein Hextastenfeld mit einer sechsstelligen Siebensegmentanzeige untergebracht ist. Dieses Tastenfeld gilt als Eingabestation und die Anzeige als eine Ausgabestation. Mit beiden kann recht einfach ein Programm eingetastet und getestet werden. Auch ein Cassetteninterface ist vorhanden, so daß der Anwender Programme und Daten auf einem billigen Massenspeicher ablegen kann.

Der KIM ist aber auch zum Anschluß an ein externes Terminal über das TTY-Interface ausgerüstet. Der Anwender kann, falls ihm eine ASCII-Datenstation zur Verfügung steht, mit Hilfe des Monitors mit dem KIM über ein Terminal kommunizieren.

Der 6530 ist ein sehr interessanter Baustein, da er viele Elemente und Funktionen in sich vereinigt.

Die Abbildung 1.1-3 zeigt seine innere Struktur.
Er besitzt ein 1k x 8bit ROM, ein 64 x 8bit RAM (als Datenspeicher), ein Ein- / Ausgaberegister A, ein Ein- / Ausgaberegister B und einen Intervalltimer.

Die Ein- / Ausgaberegister sind sehr funktionell gestaltet. Jedes Bit von ihnen kann getrennt als Eingabekanal oder als Ausgabekanal programmiert werden. Die entsprechende Programmierung wird über das zugehörige Datenrichtungsregister durchgeführt.

In den Programmbeispielen von Kapitel 4. wird hierauf noch genauer eingegangen. Bit 6 von Port B steht dem Anwender nicht zur Verfügung, es wird als CS-Eingang benutzt.

R6530 Interne Architektur

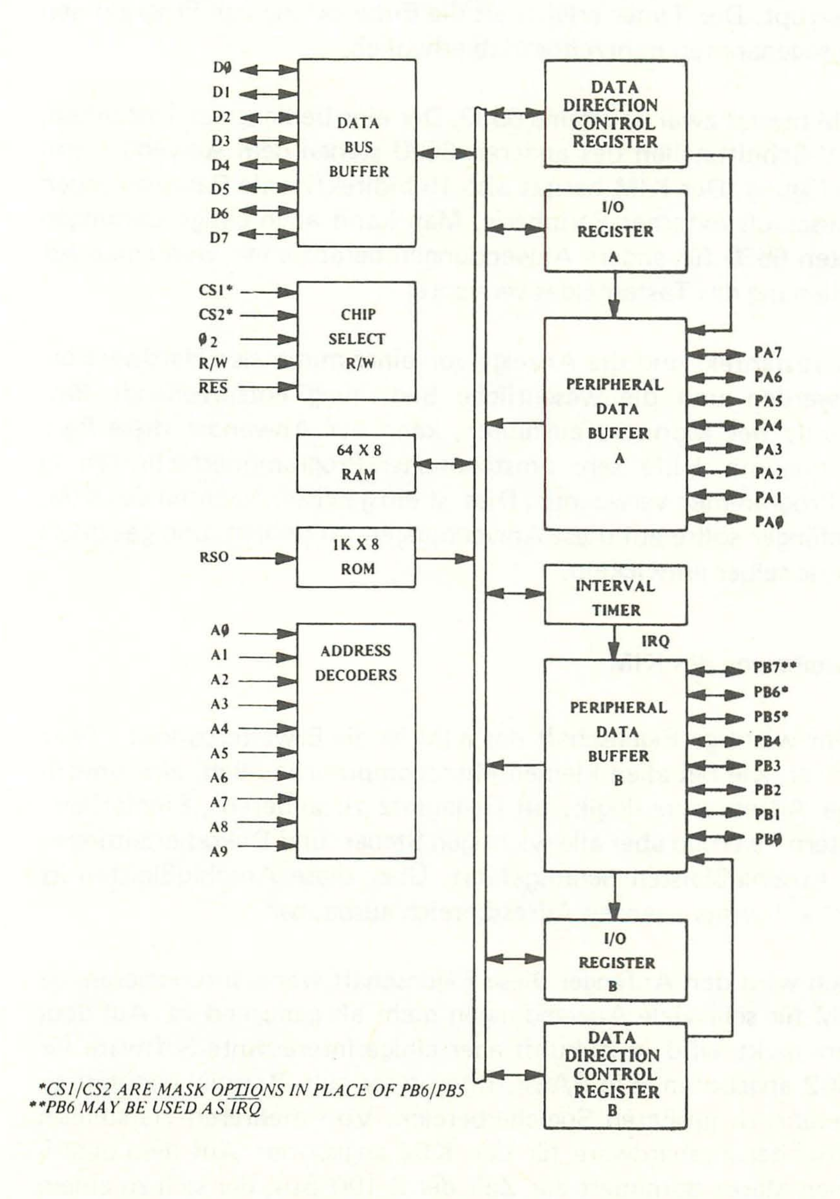


Abb.: 1.1-3

Der Intervalltimer bedarf noch besonderer Erwähnung, da er eine Besonderheit des 6530 ist. Der Timer wird von der CPU auf einen bestimmten Wert gesetzt. Er zählt dann selbständig, durch den Systemtakt gesteuert, herunter. Wenn er den Wert Null annimmt, erzeugt er ein Interrupt. Der Timer erleichtert die Entwicklung von Programmen für den sogenannten Echtzeitbetrieb erheblich.

Der KIM besitzt zwei Bausteine 6530. Der eine bedient das Tastenfeld, die TTY-Schnittstellen des anderen 6530 stehen dem Anwender voll zur Verfügung. Der KIM besitzt also 15 bidirektionale Datenleitungen zum Anschluß externer Peripherie. Man kann auch einige Leitungen des ersten 6530 für andere Anwendungen heranziehen, wenn man auf die Bedienung des Tastenfeldes verzichtet.

Da das Tastenfeld und die Anzeige von einer minimalen Hardware bedient werden und die wesentliche Bedienung entsprechende Programmteile des Monitors ausführen, kann der Anwender diese Peripherie nur mit Hilfe sehr umständlicher Programmiertechniken in seinen Programmen verwenden. Dies ist ein gewisser Nachteil des KIM. Der Anfänger sollte auf diese Anwendungen verzichten und geeignete Peripherie selber entwickeln.

Die Erweiterung des KIM

Eine sehr wichtige Eigenschaft des KIM ist die Erweiterbarkeit. Zwar hat auch er, wie bei allen kleinen Microcomputern üblich, eine unvollständige Adressierungslogik. Im Gegensatz zu anderen „Einplatinencomputern" werden aber alle wichtigen Steuer- und Dekodierleitungen an die Anschlußleisten herausgeführt. Über diese Anschlußleisten ist der KIM auf seinem ganzen Adressbereich ausbaubar.

Natürlich wird den Anfänger diese Eigenschaft wenig interessieren, da 1k RAM für sehr viele Anwendungen mehr als genügend ist. Auf dem Softwaremarkt wird in Zukunft aber einige interessante Software für den 6502 angeboten. Ein BASIC-Interpreter zum Beispiel benötigt einen wesentlich größeren Speicherbereich. Von mehreren Herstellern wird Erweiterungshardware für den KIM angeboten. Auf dem amerikanischen Markt dominiert zur Zeit der S 100 Bus, der sich zu einem gewissen Standard der Microcomputerhersteller entwickelt hat. Die

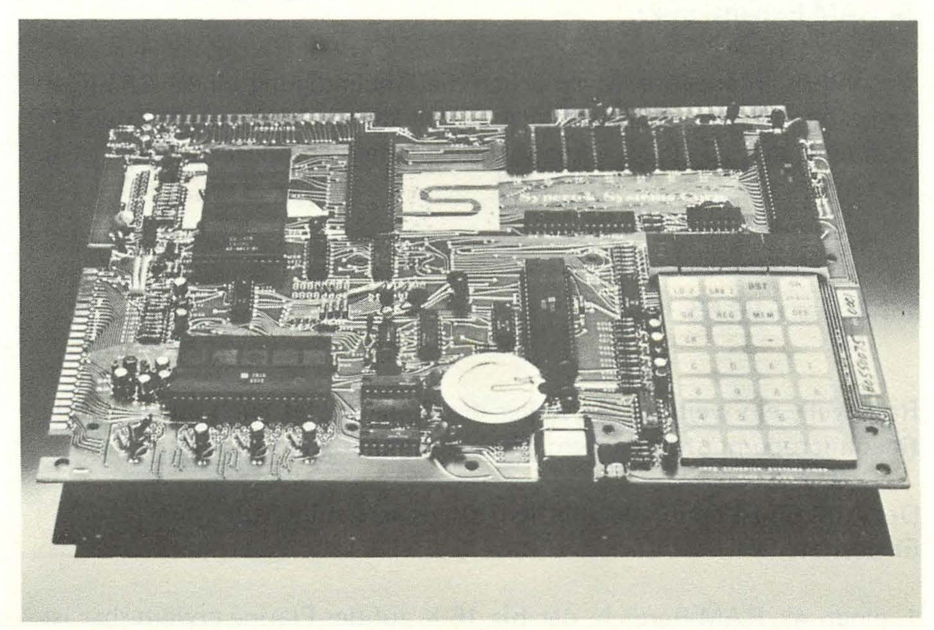
wohl preisgünstigste Hardware ist für den S 100 Bus konzipiert. Für ihn ist das Spektrum des Hardwareangebotes auch wohl am größten geworden. Der S 100 Bus ist in erster Linie für 8080 und Z 80 Computersysteme ausgelegt. Der KIM ist nicht an diesen Bus anschließbar.

Allerdings bietet ein amerikanischer Hersteller eine Erweiterungsmöglichkeit des KIM für den S 100 Bus an. (KIMSI Interface / Motherboard). Diese Erweiterungsplatine stellt einen Bus auf S 100 her und enthält die entsprechende Logik für die Normwandlung. Kann sich der KIM-Besitzer für eine Erweiterung auf S 100 Norm entschließen, steht ihm ein fast schon unübersichtliches Angebot an Zusatzhardware zur Verfügung.

1.1.2 Der VIM-1

Von Synertek, einem Hersteller des 6502, wird ein sehr interessanter Einplatinenmicrocomputer angeboten, der VIM-1.

Der VIM ist eine Erweiterung des KIM. Er besitzt einen 4k-Monitor, der in ROM' s abgelegt ist. Der RAM-Bereich umfaßt 1k byte.



Der VIM I

An Peripherie hat der VIM:

1. ein Cassettenrecorderinterface mit zwei verschiedenen Baudraten (135 Baud, zum KIM kompatibel und 1200 Baud),
2. eine TTY-20mA-Schnittstelle,
3. ein Systembuserweiterungsinterface,
4. ein TV-Controller-Interface,
5. ein CRT kompatibles Interface,
6. ein Doppelfunktionstastenfeld mit 28 Tasten,
7. eine sechsstellige Hexanzeige,
8. 15 bidirektionale Datenkanäle für externe Peripherie.

Die von Synertekangekündigten Erweiterungsmöglichkeiten des VIM sind bemerkenswert. In naher Zukunft soll der VIM um folgende Elemente erweiterbar sein.

1. TV-Interfacekarte mit ASCII-Tastenfeld,
2. BASIC-Interpreter,
3. residenter Assembler und Editor,
4. Portexpansionskit,
5. RAM Expansionskit.

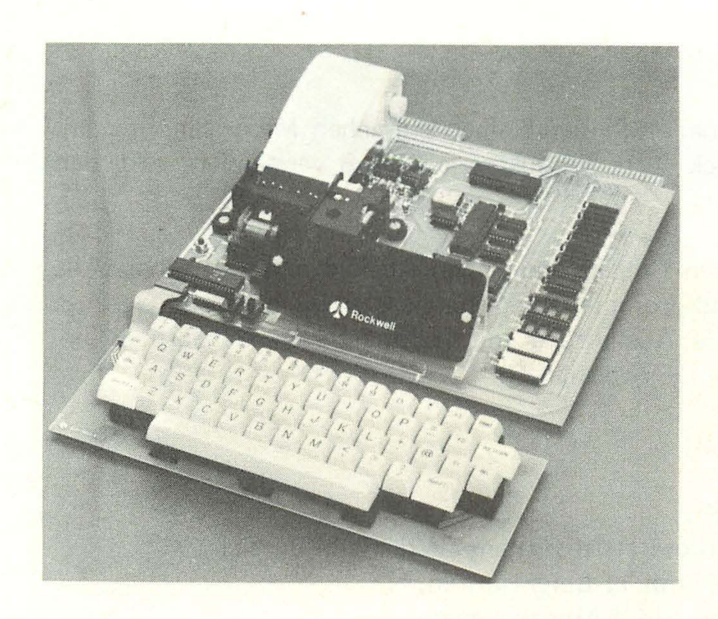
Besonders interessant ist natürlich die Ankündigung eines BASIC-Interpreters und eines Assemblers, da mit diesen Ausbaustufen der VIM sich endgültig zu einem ausgewachsenen Microcomputersystem gemausert hat.

1.1.3 Der AIM 65

Rockwell, ein weiterer Hersteller des 6502, bietet auch einen Einplatinenmicrocomputer mit dem 6502 als CPU an, den AIM 65.

Der AIM 65 ist ebenfalls sehr bemerkenswert aufgebaut.
Er besitzt:

1. einen 4k RAM-Bereich, der bis 16 K auf der Platine erweiterbar ist,
2. einen 1k RAM-Bereich, der bis auf 4K auf der Platine ausbaubar ist.



Der AIM 65

Seine Systemperipherie ist besonders interessant. Er besitzt nämlich ein ASCII-Tastenfeld und eine 20 stellige alphanumerische Anzeige. Sogar ein 20stelliger Matrixdrucker ist auf der Platine untergebracht.

Mit der sehr aufwendigen Systemsoftware wird das Arbeiten am 6502 wesentlich vereinfacht. Da der ROM-Bereich auf 16k und der RAM-Bereich auf 4K erweiterbar sind, kann ein Assembler und ein Texteditor installiert werden. Auch ein BASIC-Interpreter wird von Rockwell für den AIM 65 angeboten.

An externen Peripherieanschlußmöglichkeiten bietet der AIM 65:

1. ein TTY-20mA-Interface,
2. zwei Cassettenrecorder-Schnittstellen,
3. zwei 8bit bidirektionale Ports,
4. einenseriellen Ein- / Ausgabeport,
5. zwei 16bit Intervalltimer, die von externen Impulsen steuerbar sind.

Auch der AIM 65 ist im ausgebauten Zustand kein „kleiner" Microcomputer mehr.

1.1.4 Der Alpha 1

Die Berliner Firma MCS bietet einen einfachen Microcomputer an, der als Grundstock ihrer Produktserie Beta 8 verstanden wird, den Alpha 1.

Der Alpha 1 ist in drei verschiedenen Ausführungen lieferbar. Alle Ausführungen sind komplett und betriebsfertig in einem Gehäuse untergebracht. Die Ausführung mit transparentem Gehäuse ist besonders gut für die Lehre geeignet.

Der Alpha 1 besitzt:

1. den 6502 als CPU,
2. ein leistungsstarkes Betriebssystem im 2k ROM-Bereich,
3. ein Hextastenfeld mit 11 Befehlstasten,
4. 16 bidirektionale Ein- / Ausgabekanäle,
5. eine V24 und 20mA Schnittstelle für ein Terminal,
6. zwei Cassettenrecorderinterfaces
7. eingebautes Netzgerät.

Besonders soll hervorgehoben werden, daß der Alpha 1 unbeschränkt und ohne zusätzliche Lötarbeiten mit Hilfe der Steckkarten der Serie Beta 8 erweiterbar ist.

1.2 Tischcomputer

Die hier vorgestellten zwei Microcomputer unterscheiden sich von den im Abschnitt 1.1 besprochenen in folgenden zwei Punkten.

1. Sie werden in einer größeren Ausbaustufe angeboten. Beide haben einen, in einem ROM abgelegten BASIC-Interpreter. Sie sind also in erster Linie für Anwender gedacht, die nicht auf der Maschinenebene programmieren wollen.

2. Sie sind in einem Gehäuse mit Stromversorgung untergebracht, getestet und geprüft. Der Anwender muß also keine Zeit für Aufbauarbeiten verwenden. Daher werden sie auch in die Gruppe der Tischcomputer eingeordnet.

Diese Tischcomputer werden eher den Anwender ansprechen, der sich einen fertigen, einsatzbereiten Computer mit einer möglichst leistungsfähigen Software wünscht. Natürlich muß er gewisse Nachteile hinnehmen. Diese Computer sind, da sie möglichst kostengünstig produziert werden, nicht so leicht ausbaubar, wie die Computer des 1. Abschnittes. Die Ausbaubarkeit bleibt auf die vom Hersteller angebotenen Elemente beschränkt. Beide haben nur wenige Ein- / Ausgabekanäle. Der Anwender ist also beschränkt in den Möglichkeiten, externe Peripherie anzuschließen. Dies sind natürlich keine gravierenden Fehler, da durch entsprechende Umbauarbeiten an den Geräten auch hier Abhilfe geschaffen werden kann.

Erstaunlich ist bei beiden Geräten der relativ niedrige Preis bei der sehr hohen Leistungsfähigkeit der Geräte.

Sie sind die idealen Helfer für denjenigen, der im Beruf oder im privaten Bereich typische Aufgaben und Probleme für eine EDV-Anlage zu bewältigen hat.

Zum Beispiel:
1. Steuerberechnungen,
2. Abrechnungen,
3. Lohn und Buchhaltung (natürlich nur mit Einschränkungen),
4. Berechnungen von Abschreibungen,
5. Datenanalyse,
6. graphische Deutung von Daten.

Vor allem im Bereich der Ausbildung könnten sie sicherlich teure Timesharing – Anlagen ersetzen, andererseits vielen Schulen und Instituten den Einstieg in das Unterrichten der Programmierungsverfahren erst ermöglichen.

1.2.1 Der Apple II

Die CPU des Apple II ist ein 6502. In der einfachsten Ausbaustufe besitzt der Apple einen 4K RAM-Bereich. Dieser ist bis auf 48K ausbaubar. Der ROM-Bereich umfaßt 8K byte. Auch dieser kann um zwei zusätzliche Bausteine, wie vom Hersteller angekündigt wird, erweitert werden. Ein ASCII-Tastenfeld ist die systeminterne Eingabestation. Im ROM-Bereich sind untergebracht:

1. ein 2K Monitor,
2. ein 6K BASIC-Interpreter, der vor allem auf eine einfache Programmierung des graphischen TV-Interfaces ausgelegt ist.

An Schnittstellen für externe Peripherie sind vorhanden:

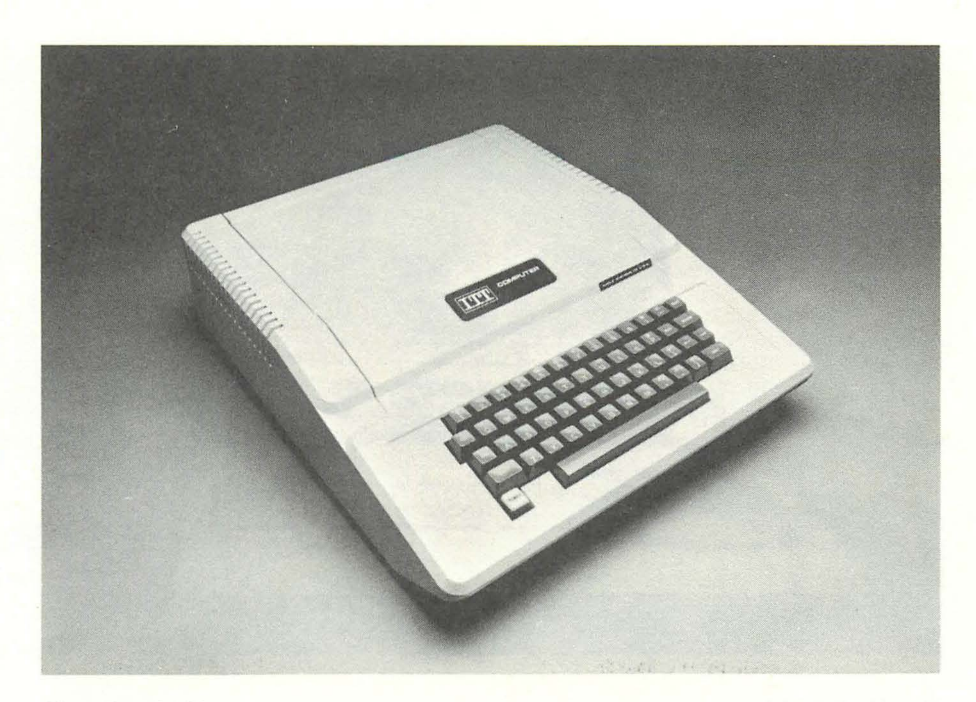
1. Ein TV-Video-Interface für Farbfernseher. Hierzu ist ein Anschlußboard für PAL-Fernseher lieferbar.
2. Ein Anschlußboard für einen Drucker ist lieferbar.
3. Ein schnelles Interface für einen Cassettenrecorder mit einer Übertragungsrate von 1500 Baud ist eingebaut.
4. Ein Anschlußverbinder mit vollständig gepufferten Bus-Leitungen ist vorhanden.
5. 4 Input- und 4 Output-Kanäle stehen zur Verfügung.

Der 2K Monitor ist recht aufwendig und bietet under anderem:

1. volle Cursorcontrolle (für das Videointerface).
2. Einzelschritt- und Tracebetrieb,
3. einen softwaresimulierten 16 bit Prozessor,
4. einen Disassembler und einen Miniassembler,
5. Floatingpoint – Unterroutinen.

Obwohl der BASIC-Interpreter wenig mathematische Anwendungen bietet, ist er sehr interessant. Er bietet an Besonderheiten:

1. umfangreiche Möglichkeiten für die Textverarbeitung,
2. Mehrfachanweisungen pro Programmzeile,
3. einfache Programmierung von farbigen Graphiken,
4. Zugriff auf maschinengeschriebene Unterprogramme.



Der Apple II

1.2.2 Der PET

Commodore bietet nun seinen PET an (personal electronic transactor Model 2001).
Der PET ist ein leistungsfähiger Tischcomputer. Er wird sicherlich viele Interessenten finden, da er ein abgerundetes System darstellt, dem nur noch ein Drucker fehlt.

Die CPU des PET ist der 6502. Der PET ist ausgerüstet mit einem Spezialtastenfeld, einer Cassettenstation und SW-Monitor.
Der Einplatinencomputer besitzt:

1. einen 14K ROM-Bereich,
2. einen 4K RAM-Bereich, der intern um 4K erweiterbar ist. Eine externe Erweiterung um weitere 24K ist möglich.
3. Ein IEEE–488–Interface ermöglicht den Anschluß externer, intelligenter Peripherie.



Der PET 2001

Der ROM-Bereich besitzt:

1. einen 1K Monitor,
2. ein 4K Betriebssystem, das die leichte Programmierung von Graphiken ermöglicht,
3. einen 8K BASIC-Interpreter.

Der sehr schnelle BASIC-Interpreter bietet:

1. Textverarbeitungsmöglichkeiten,
2. eine 10stellige Fließkommazahlendarstellung,
3. die wichtigsten mathematischen Funktionen,
4. den Zugriff auf maschinengeschriebene Unterprogramme.

1.3 Microcomputerentwicklungssysteme

Die im Abschnitt 1.1 beschriebenen kleinen Microcomputer bieten nur in beschränktem Umfang dem Anwender die Möglichkeit, größere Anwenderprogramme für ein 6502-Microcomputersystem zu entwickeln. Es fehlen zum Beispiel PROM-Programmierer und schnelle Massenspeicher.

Der schon traditionelle Weg der Systementwicklung ging über eine Großrechenanlage, in der ein Cross Assembler installiert wird. Für Anwender, die nicht auf eine eigene Großrechenanlage zurückgreifen können, sind die Entwicklungskosten sehr erheblich, da sie die teuren Rechenzeiten an einer Großrechenanlage einkalkulieren müssen. Einige Hersteller von Microcomputern bieten daher Microcomputerentwicklungssysteme an, die die Ausbaustufe einer kleinen Rechenanlage der mittleren Datentechnik erreichen können.

1.3.1 Das System 65

Rockwell bietet für die 6500-Serie das Entwicklungssystem „System 65" an.

1. den 6502 als CPU,
2. einen 14K ROM-Bereich mit der gesamten Systemsoftware,
3. mehrere RAM-Module. (Ein Modul besitzt 16K RAM).
4. Input-/Outputmodule,
5. einen PROM-Programmierer,
6. zwei Minifloppylaufwerke als schnelle Massenspeicher,
7. einen Matrixdrucker,
8. ein CRT-Terminal.

Die Software besteht aus einem Texteditor, einem Assembler und Debug-Programmen.
Der Texteditor ermöglicht die Onlineentwicklung von Programmen im Assemblercode. Der Assembler übersetzt sie dann in den Maschinencode, und mit Hilfe der Debug-Programme kann die Software getestet werden.

1.3.2 Der Beta 8

Die Firma CMS bietet außer dem Alpha 1 ein umfangreiches Programm auf der Basis des 6502 an, das System Beta 8.

Der Anwender kann aus dieser Palette zum Beispiel ein Entwicklungssystem nach seinen eigenen Vorstellungen aufbauen. Alle Bausteine des Beta 8 sind auf Europakarten aufgebaut.

Das Programm Beta 8 umfaßt unter anderem:

1. CPU-Karte,
2. Monitorkarte,
3. EPROM-Programmiereinheit,
4. BUS-Kontrollkarte,
5. 8K Assembler und Editor,
6. RAM-Karten,
7. I/O-Karten,
8. Minifloppydoppellaufwerk.

An Entwicklungssoftware stehen zur Verfügung:

ein 6K Assembler (2-Pass-Assembler),
ein 2K Editor (sehr leistungsstark),
ein 2K File- Verwaltungsprogrammsystem,
ein 2K Monitor- und Debug-Programm.

2. Die Software

Software ist ein englischer Fachausdruck, den man nicht einfach durch den Begriff Programm ersetzen kann. Unter Software versteht man nämlich die Menge aller denkbaren und möglichen Programme einer Rechenanlage. Software und Hardware zusammen ergeben erst den leistungsfähigen Computer

Meistens wird für die Softwareentwicklung mehr Zeit benötigt als für die Entwicklung der Hardware. Oft ist die Software eines Computers teurer als seine Hardware. Auf dem Mini- und Microcomputersektor wird dies besonders deutlich spürbar.

Was ist nun ein Programm?
Soll ein Computer eine bestimmte Aufgabe erfüllen, so muß ihm dies in einer ihm verständlichen Form mitgeteilt werden. Die CPU eines Computers versteht nur einzelne Befehle. Werden nun eine bestimmte Anzahl von Befehlen nacheinander ausgeführt, so kann damit die gestellte Aufgabe erfüllt werden. Eine Aneinanderreihung von Befehlen nennen wir Programm.

Je nach Aufgabenstellung kann man die Programme eines Computers in Betriebsprogramme und Anwenderprogramme einteilen. Beide Gruppen werden in Kapitel 2.2 und 2.3 genauer vorgestellt.

Die Form, wie ein Programm dem Computer mitgeteilt wird, nennt man Programmiersprache. Man kann die Programmiersprachen wiederum in zwei Gruppen einteilen.

Die erste Gruppe sind die maschinennahen Sprachen. Bei diesen Sprachen entspricht eine einzelne Anweisung genau einem einzelnen Maschinenbefehl der CPU.

Die andere Gruppe sind die höheren Programmiersprachen, man nennt sie auch problemorientierte Sprachen. Diese Sprachen erleichtern die Programmierung bestimmter, zum Beispiel mathematischer Probleme (BASIC und FORTRAN u. a.). Eine einzelne Programmanweisung entspricht bei diesen Sprachen oft einer sehr komplexen Verknüpfung von vielen Maschinenbefehlen.

Die maschinennahen Sprachen werden auch Assemblersprachen genannt. Da in der Literatur der Begriff Assembler nicht einheitlich verwendet wird, wollen wir hier nun einige Begriffe festlegen.

Eine jede CPU (natürlich auch Microprozessor) besitzt einen bestimmten durch die Hardware festgelegten Befehlssatz (instruction set). Unter einem Befehl wird ein bestimmtes Bitmuster, Operationscode genannt, verstanden, das eine bestimmte Ablaufsteuerung in der CPU bedingt.

Dieses Bitmuster (oder Operationscode) kann man natürlich direkt in den RAM-Speicher laden. Ein Programm, in dieser Form dargestellt, heißt im Maschinencode dargestellt. Diese Programmiersprachen, die der Maschine am nächsten sind, nennt man Maschinencode.

Für den Programmierer bedeutet es eine große Erleichterung, wenn er nicht in Bitmustern, deren Bedeutung man sich schlecht merken kann, sondern in Buchstabenmustern oder Zeichenketten programmieren kann. Jedem Maschinenbefehl wird eine bestimmte Zeichenfolge zugeordnet (z. B. LDA oder STA). Ein Programm, das in solchen Zeichenketten (Worten) formuliert ist, kann die CPU nicht mehr direkt verstehen, es muß in den Maschinencode übersetzt werden. Diese Übersetzung kann der Programmierer selber durchführen, oder man verwendet ein geeignetes Übersetzungsprogramm.

Solche Programme können in großen Rechenanlagen installiert sein, dann nennt man sie Cross Assembler, oder aber sie laufen im Microcomputer selber ab, dann heißen sie Assembler.

Ein Programm, daß in der leichter verständlichen Form dargestellt wird, nennen wir Programm im Assemblercode. Diese Programmiersprache heißt also Assemblercode.

2.1 Die Grundlagen

Für manche Anwender ist der Einstieg in die Microcomputer-Technologie sehr schwierig, da ihnen das sehr umfangreiche Basiswissen fehlt. Die Grundlagen der Digitalelektronik sind zwar eine unabdingbare Voraussetzung, reichen aber nicht zum Verständnis eines Microcomputers aus. Hier sollen nun einige Grundlagen geboten werden, die unbedingt für das Verständnis dieses Buches notwendig sind. Dieses Kapitel kann nicht erschöpfend sein, da ein Microcomputer ein sehr komplexes Gerät ist. Will der Leser ein tieferes Verständnis der Microcomputertechnik gewinnen, muß er auf die entsprechenden Fachbücher der verschiedenen Hersteller verwiesen werden.

Die Kommunikation zwischen dem Microprozessor (der CPU) und der Peripherie (ROM-, RAM- Bereich, Schnittstellen) geschieht über ein BUS-System. Das Bussystem des 6502 besteht aus 16 Adressleitungen (dem Adressbus), 8 Datenleitungen (dem Datenbus), und etwa 10 Steuerleitungen (dem Steuerbus). Im Gegensatz zur Analogtechnik werden in der Datentechnik pro Leitung nur zwei Informationszustände übertragen. Diese Informationszustände werden durch zwei Zeichen gekennzeichnet, 0 und 1. 0 entspricht ca. 0 V, 1 entspricht in der TTL-Logik ca. 5 V. In der englischen Fachliteratur werden die Zeichen L (low) und H (high) verwendet. Auch in vielen deutschen Fachbüchern ist diese Bezeichnung zu finden. Wir wollen aber die Ziffern zur Bezeichnung verwenden, da die Daten meist als Zahlen zu interpretieren sind.

Diese kleinste Informationseinheit, die zwei Zustände annehmen kann, bezeichnen wir als Bit (als Einheit: bit).

Die nächst höhere Einheit der Information ist ein Wort (1 byte). Ein Byte umfaßt 8 bit (1 byte = 8 bit). Im Gegensatz zum Begriff Byte, der eine 8 bit Länge hat, verwenden wir den deutschen Begriff Wort in allgemeinerer Bedeutung. Das Adresswort des 6502 besteht zum Beispiel aus 16 Bits oder 2 Bytes.

Der Adressbus des 6502 ist unidirektional, das heißt nur in eine Richtung orientiert. Die Adressworte werden immer von der CPU an die Peripherie gesendet.

Der Datenbus ist bidirektional, das heißt in zwei Richtungen orientiert. Nicht nur die CPU sendet Datenworte an die Peripherie, sondern auch die Peripherie sendet Datenworte über die gleichen Leitungen an die CPU. Diese Vorgänge müssen natürlich exakt gesteuert werden. Die entsprechenden Steuersignale (Steuerworte) werden von der CPU über den Steuerbus gesendet.

Der Steuerbus ist unidirektional, in eine Richtung orientiert. Die meisten Steuerleitungen werden nur von der CPU mit Information versehen. Einige Leitungen senden aber auch Informationen von der Peripherie an die CPU. Werden langsame Speichersysteme eingesetzt, muß zum Beispiel über eine entsprechende Steuerleitung das Ende einer Leseoperation der CPU mitgeteilt werden.

2.1.1 Die mathematischen Grundlagen

Damit dem Leser einsichtig wird, daß man ein Datenwort, ein Adresswort oder auch ein Steuerwort durch eine Zahl darstellen kann, muß ein kleiner Exkurs in die Mathematik entwickelt werden.

2.1.1.1 Die Zahlensysteme

1. Das Dezimalsystem

Jedem Leser ist das dezimale Zahlensystem geläufig. Dennoch wollen wir uns dazu einige Gedanken machen, die uns beim Verständnis anderer Systeme behilflich sind.

Die Zahl 248 besteht aus drei Ziffern. Die Ziffern sind aus der Ziffernmenge [0, 1, 2, 3, ..., 9] entnommen. Wir sagen, die Zahl 248 besitzt 2 Hunderter, 4 Zehner und 8 Einer.

$$248 = 2 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$$

Man sagt: Das Dezimalsystem besitzt die Basis 10. Wenn Verwechslungen mit anderen Systemen möglich sind, schreiben wir: 248_{10}.

Wenn man etwas genauer hinsieht, erkennt man, daß das Dezimalsystem in gewisser Hinsicht wilkürlich ist. Historisch gesehen stimmt das

auch. Es gibt und gab Zahlensysteme mit anderen Basen. Man denke nur an die angelsächsischen Maßeinheiten. Wir wollen hier nur drei weitere Zahlensysteme kennenlernen, die in die Microcomputerprogrammierung Eingang gefunden haben.

2. Das Dualsystem (2er- System)

Das einfachste System, mit der niedrigsten Basis, ist das System zur Basis 2, das Dualsystem.

Das Dualsystem besitzt nur zwei Ziffern. Die Ziffernmenge: [0, 1].
10110101_2 ist eine achtstellige Dualzahl. Die Ziffernfolge ist durch die folgende Gleichung definiert.

$$10110101_2 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 1 \times 128 + 0 \times 64 + 1 \times 32 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$$

Das Dualsystem ist einfach aufgebaut. Große Zahlen besitzen allerdings viele Stellen. Daher ist es etwas unübersichtlich.

Da ein Bit zwei Zustände annehmen kann und wir diese auch durch die Ziffern 0 und 1 kennzeichnen, kann ein Bit als eine Ziffer einer Dualzahl verstanden werden. Ein 8 bit Datenwort ist als achtstellige Dualzahl interpretierbar. Ein 16 bit Adresswort stellt eine 16stellige Dualzahl dar.

3. Das Oktalsystem (8er-System)

Verwenden wir die Basis 8, so wird eine Zahl im Oktalsystem dargestellt.
Das Oktalsystem besitzt die Ziffernmenge [0, 1, 2, 3, 4, 5, 6, 7].
Eine vierstellige Oktalzahl wird wie folgt definiert:

$$2463_8 = 2 \times 8^3 + 4 \times 8^2 + 6 \times 8^1 + 3 \times 8^0$$
$$= 5 \times 512 + 4 \times 64 + 6 \times 8 + 3 \times 1$$

Da 8 eine 2er Potenz ist, $8 = 2^3$, ist der Zusammenhang zwischen dem Oktalsystem und dem Dualsystem denkbar einfach zu finden. Dies

sei an einem einfachen Beispiel erläutert.

$$
\begin{aligned}
237_8 &= 2 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 \\
&= 2 \times 2^6 + 3 \times 2^3 + (4 + 2 + 1)\, 2^0 \\
&= 2^7 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 \\
&= 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 \\
&\quad + 1 \times 2^1 + 1 \times 2^0 \\
&= 10011111_2
\end{aligned}
$$

Das Verfahren kann abgekürzt werden. Die Ziffern des Oktalsystems interpretieren wir als Dualzahlen. Wenn wir diese dann stellenrichtig einsetzen, steht die Dualzahl schon da.

Oktalziffer	Dualziffer
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

$$2\,3\,7_8 = 010\ 011\ 111_2$$

Die umgekehrte Herleitung geht besonders einfach. Wir fassen die Ziffern der Dualzahl immer in Dreiergruppen zusammen und wandeln diese in die Oktalziffern um.

$$10011111_2 = 010\ 011\ 111 = 237_8$$

Da zwischen Dualzahlen und Oktalzahlen ein so einfacher Zusammenhang besteht, werden die Worte oft als Oktalzahlen dargestellt. Diese Darstellung hat den Vorteil der besseren Übersichtlichkeit.

4. Das Hexadezimalsystem (Hexalsystem, 16er System)

Eine weitere Verbreitung als das Oktalsystem hat aber das Hexadezimalsystem gefunden, obwohl es etwas unangenehmer ist, da der

Mensch nur das Rechnen mit numerischen Ziffern gewohnt ist. Das Hexadezimalsystem hat aber den eindeutigen Vorteil der Kurzdarstellung. Ein Byte wird nämlich durch nur zwei Ziffern dargestellt.

Das Hexadezimalsystem besitzt als Basis die Zahl 16. Daher benötigt man zur Darstellung einer Zahl 16 verschiedene Ziffern. Die zehn Ziffern des Dezimalsystems müssen um weitere sechs Ziffern erweitert werden. Es hat sich eingebürgert, hierfür die ersten sechs Buchstaben des Alphabetes zu verwenden.

Die Ziffernmenge: [0, 1, 2, ... 9, A, B, C, D, E, F].

Eine vierstellige Hexalzahl wird wie folgt definiert.

$$3CA7_{16} = 3 \times 16^3 + 12 \times 16^2 + 10 \times 16^1 + 7 \times 16^0$$
$$= 3 \times 4096 + 12 \times 256 + 10 \times 16 + 7 \times 1$$

Da auch 16 eine 2er Potenz ist, $16 = 2^4$, kann eine Zahl vom Hexalsystem in das Dualsystem und umgekehrt leicht umgewandelt werden.

Ein Beispiel:

$$AD_{16} = 10 \times 16^1 + 13 \times 16^0$$
$$= (8 + 2) \times 2^4 + (8 + 4 + 1) \times 1 \times 2^0$$
$$= 8 \times 2^4 + 2 \times 2^4 + 2^3 + 2^2 + 1 \times 2^0$$
$$= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 10101101_2$$

Das Verfahren kann auch abgekürzt werden, wenn wir die Ziffern des Hexalsystems durch Dualzahlen ersetzen.

Hexziffer	dual	dezimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

$AD_{16} = 1010\ 1101_2$

Die Umrechnung vom Dualsystem in das Hexalsystem ist denkbar einfach. Wir teilen die Dualzahl in Gruppen zu vier Stellen auf. Jede Gruppe stellt eine Hexziffer dar.

$11\ 0010\ 1100\ 0100_2 = 3204_{16}$

2.1.1.2 Umrechnungen

Es wurde schon der Zusammenhang zwischen dem Dualsystem und dem Oktalsystem sowie dem Hexalsystem erläutert.

Die Umformungen vom Oktal- nach Hexalsystem und zurück wird man sinnvoll über das Dualsystem durchführen. Dieser Weg ist besonders einfach.

Zwei Beispiele:

$$\begin{aligned} 37AB_{16} &= 0011\ 0111\ 1010\ 1011_2 \\ &= 011\ 011\ 110\ 101\ 011_2 \\ &= 33653_8 \end{aligned}$$

$46352_8 = 100\ 110\ 011\ 101\ 010_2$
$= 0100\ 1100\ 1110\ 1010_2$
$= 4CEA_{16}$

Ebenso wird man die Wandlung zwischen dem Dezimalsystem und den anderen Systemen immer auf eine Wandlung zwischen dem Dezimal- und dem Dualsystem zurückführen. Diesen Weg wollen wir hier beschreiten.

1. Die Wandlung vom Dualsystem in das Dezimalsystem

Dieser Weg ist sehr einfach. Wir müssen nur die Potenzen von 2 kennen und addieren.

$$10111011_2 = 2^7 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0$$
$$= 128 + 32 + 16 + 8+2 + 1$$
$$= 187_{10}$$

2. Die Wandlung vom Dezimalsystem in das Dualsystem

Die entsprechende Umrechnungsvorschrift entspricht der Division mit Rest. An einem Beispiel sei dies erläutert. Wir wollen 207_{10} umrechnen.

207 : 2 = 103 Rest 1
103 : 2 = 51 Rest 1
51 : 2 = 25 Rest 1
25 : 2 = 12 Rest 1
12 : 2 = 6 Rest 0
6 : 2 = 3 Rest 0
3 : 2 = 1 Rest 1
1 : 2 = 0 Rest 1

Daraus folgt: $207_{10} = 11001111_2$

Die Reste werden als die Ziffern der Dualzahl verstanden. Dies können wir uns auch erklären.

$z_0, z_1, ..., z_7$ seien die Ziffern der Dualzahl. Dann gilt:

$$207_{10} = z_7 \times 2^7 + z_6 \times 2^6 + z_5 \times 2^5 + z_4 \times 2^4 + z_3 \times 2^3 + z_2 \times 2^2 + z_1 \times 2^1 + z_0 \times 2^0$$

$$= z_0 + 2 \times (z_1 + 2 \times (z_2 + 2 \times (z_3 + 2 \times (z_4 + 2 \times (z_5 + 2 \times (z_6 + 2 \times z_7))))))$$

2.1.1.3 Das Rechnen mit Dualzahlen

Natürlich kann man die vier Grundrechenarten in beliebigen Zahlensystemen durchführen. Wir wollen uns hier nur auf die Betrachtung der Grundrechenarten im Dualsystem beschränken.

1. Addition

Die Addition zweier einstelliger Dualzahlen wird durch die folgende Additionstafel definiert:

	+	0	1	1. Summand
	0	0	1	
2. Summand	1	1	0	mit Übertrag

Die Addition mehrstelliger Zahlen wird stellenweise durchgeführt mit Beachtung des Übertrages.

	10110100
	11011011
Übertrag	1111
Summe:	110001111

2. Die Subtraktion

Die Subtraktion kann auf zwei verschiedene Arten definiert werden.

1. Art: Subtraktion durch „Borgen einer 1"

Minuend	10110100
Subtrahend	01111101
geborgt	11111111
Differenz	00110111

2. Art: Subtraktion durch Addition des 2er Komplementes

Die Subtraktion kann auf die Addition zurückgeführt werden. Dazu muß man sich aber auf einen begrenzten Zahlenbereich beschränken. Da alle Computer nur einen begrenzten Zahlenbereich besitzen, ist dies für unsere Anwendungen keine Einschränkung.
Wir legen unseren Zahlenbereich auf acht Stellen fest. Treten größere Zahlen auf, werden die überzähligen Ziffern einfach vergessen.

Der Subtrahend sei wieder 0111 1101. Von diesem Subtrahenden bilden wir nun das Komplement, indem wir die 1 durch eine 0 und die 0 durch eine 1 ersetzen (stellenweise). Anschließend addieren wir zum Komplement eine 1 und erhalten so das 2er Komplement des Subtrahenden.

```
0111 1101
1000 0010     Komplement
      + 1
1000 0011     2er Komplement
```

Das 2er Komplement wird nun zum Minuenden hinzuaddiert.

```
           1011 0100
           1000 0011
Übertrag  1
          1]1011 0111
```

Die 9. Stelle der Summe wird vergessen. Es steht wieder eine korrekte Differenz da.

Ein weiteres Beispiel:

```
1100 1011 – 1001 110

1001 1110
0110 0001     Das Komplement
       + 1
0110 0010     2er Komplement
```

```
  11001011
  01100010
 11    1
─────────────
1]00101101
```

1100 1011 – 1001 1110 = 101101

Diese Rechnung prüfen wir in dezimaler Darstellung nach:

Minuend: $11001011_2 = 128 + 64 + 8+2 + 1 \quad = 203_{10}$

Subtrahend: $1001\ 1110_2 = 128 + 16 + 8+4 + 2 \quad = 158_{10}$

45_{10}

Differenz: $101101_2 = 32 + 8 + 4 + 1 = 45_{10}$

Die Multiplikation und Division von mehrstelligen Dualzahlen werden in den Programmbeispielen von Kapitel 4.1.1 erläutert. Daher wollen wir hier nicht näher darauf eingehen.

2.1.2 Die Darstellung von Daten

Die Daten, die im Datenspeicher (RAM-Bereich) eines Computers abgespeichert werden, liegen dort in einer verschlüsselten (codierten) Form vor.

Unter einem Code versteht man eine Zuordnungsvorschrift, die die Zeichen verschiedener Zeichenmengen einander zuordnet. Bei Computern ist die eine Menge die Menge der Dualzahlen, die durch ein Datenwort darstellbar sind, und die andere Menge ein beliebiger Zeichenvorrat. Dieser Zeichenvorrat kann eine bestimmte Zahlenmenge (numerische Daten), das Alphabet oder Bildinformation u. a. m. (nicht-numerische Daten) sein.

Da die meisten Microcomputer eine Wortlänge von 8 bit haben, wollen wir uns auch hier auf eine solche beschränken.

2.1.2.1 Die Darstellung numerischer Daten.

1. Der Dualcode

Dies ist der natürlichste Code. Die Zahlen werden in das Dualsystem umgerechnet. Jeder Ziffer wird dann ein Bit des Datenwortes zugeordnet.

2. Der BCD-Code

Nicht immer ist die Darstellung im Dualcode geschickt, da die Umrechnungen doch erheblich sind. So hat man einen weiteren Code entwickelt.

Bei diesem Code werden die Ziffern einer Dezimalzahl isoliert in eine Dualzahl umgerechnet. Das ergibt dann eine Ziffernlänge von 4 bit. Diese Ziffern werden stellenrichtig abgespeichert.
In einem Byte befinden sich zwei BCD-Ziffern.

Ein Beispiel: $245_{10} = 0010\ 0100\ 0101_{BCD}$

3. Weitere Verschlüsselungen

Es gibt nicht nur den Dualcode, der ja auch beim BCD-Code die Zifferncodierung bestimmt. Bei manchen Anwendungen treten relativ häufig Störungen auf. Bei dual verschlüsselten BCD-Ziffern sind Fehler nur erkennbar, wenn zufällig eine Ziffer größer als 9 wird.

Es gibt nun einige Code, die zur Fehlererkennung und Fehlerkorrektur günstiger sind. Eine Faustregel heißt: Je mehr Bits zur Darstellung herangezogen werden, um so größer ist die Fehlererkennungswahrscheinlichkeit.

In der Tabelle 2.1.2.1 sind drei weitere Beispiele angegeben.

ASCII CODE

b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b4	b3	b2	b1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

Tabelle 2.1.2.2

Tabelle: 2.1.2.1

Ziffer	Gray-Code	2 aus 5	Exzeß-3-Code
0	0000	00011	0011
1	0001	00101	0100
2	0011	00110	0101
3	0010	01001	0110
4	0110	01010	0111
5	0111	01100	1000
6	0101	10001	1001
7	1101	10010	1010
8	1100	10100	1011
9	1000	11000	1100

2.1.2.2 Die Darstellung nichtnumerischer Daten

Ein 8 bit Wort kann 256 unterschiedliche Zeichen darstellen, natürlich nicht nur Ziffern. Ein Zeichenvorrat, der Buchstaben, Ziffern und Sonderzeichen umfaßt, heißt alphanumerischer Zeichenvorrat. Für die Verschlüsselung alphanumerischer Zeichen in die Dualdarstellungen gibt es sehr unterschiedliche Code. Zwei gebräuchliche wollen wir hier vorstellen, den ASCII-Code und den 5-Kanal-Fernschreibcode.

Der ASCII-Code (Tabelle 2.1.2.2) ist ein 8bit Code. Das 8. Bit ist allerdings ein Paritätsbit, das zur Fehlerüberwachung dient. Bei vielen Microcomputersystemen wird daher auf dieses Bit verzichtet. Die meisten Systeme arbeiten also mit einem 7 bit ASCII-Code.

Der Fernschreibcode (Abb. 2.1.2.3) ist ein 5 bit Code. Eigentlich kann man mit 5 Bits nur 32 verschiedene Zeichen verschlüsseln. Durch eine Einteilung der Zeichen in zwei Gruppen, die Zifferngruppe und die Buchstabengruppe, wird der Zeichenvorrat des Codes erhöht. Wenn ein Gruppenwechsel stattfindet, muß vorher das entsprechende Sonderzeichen gesendet werden.

Abb. 2.1.2.3

Stellenzahl	Buchstaben	Ziffern/Zeichen
1 2 3 4 5	A	CCITT
1 1 0 0 0	A	–
1 0 0 1 1	B	?
0 1 1 1 0	C	:
1 0 0 1 0	D	wer da?
1 0 0 0 0	E	3
1 0 1 1 0	F	n. f.
0 1 0 1 1	G	n. f.
0 0 1 0 1	H	n. f.
0 1 1 0 0	I	8
1 1 0 1 0	J	Klingel
1 1 1 1 0	K	(
0 1 0 0 1	L	)
0 0 1 1 1	M	.
0 0 1 1 0	N	,
0 0 0 1 1	O	9
0 1 1 0 1	P	0
1 1 1 0 1	Q	1
0 1 0 1 0	R	4
1 0 1 0 0	S	'
0 0 0 0 1	T	5
1 1 1 0 0	U	7
0 1 1 1 1	V	=
1 1 0 0 1	W	2
1 0 1 1 1	X	/
1 0 1 0 1	Y	6
1 0 0 0 1	Z	+
0 0 0 1 0	Wagenrücklauf	
0 1 0 0 0	Zeilenvorschub	
1 1 1 1 1	Buchstaben	
1 1 0 1 1	Ziffern/ Zeichen	
0 0 1 0 0	Zwischenraum	
0 0 0 0 0	ignoriere	

2.2 Betriebsprogramme

Unter einem Betriebsprogramm (auch Monitor genannt) eines Microcomputers versteht man ein Programm, das das Arbeiten mit dem Microcomputer überhaupt erst ermöglicht und diese Arbeit erleichtert. Der Monitor ist in den allermeisten Fällen in Festwertspeichern (ROMs) gespeichert und so gegen eine Zerstörung gesichert.

Die Aufgaben eines Monitors seien hier angegeben. Anschließend werden kurz die Funktionen des Monitors eines typischen, kleinen Microcomputers (KIM) vorgestellt.

Die Aufgaben eines Monitors:

1. Setzen eines definierten Anfangszustandes,
2. Aufbau einer einfachen Speicheranordnung (Struktur, Stack),
3. Durchführung einfacher Operationen wie:
   1. Lesen und Abändern einer Speicherzelle,
   2. Lesen und Abändern der Registerinhalte der CPU (central processing unit),
   3. Setzen des Programmzählers (PC), um den Ablauf eines Programmes an einer beliebigen Speicherstelle zu ermöglichen,
4. Erleichterung der Fehlersuche (debugging) durch:
   1. Einzelschrittsteuerung,
   2. Setzen von Unterbrechungen,
5. Bedienung der Peripherie, z. B.:
   1. Eingabe, Ausgabeprogramme,
   2. Programme zur Bedienung eines Cassettenrecorders.

Ein Beispiel:

Wird die Stromversorgung des KIM eingeschaltet, so muß zusätzlich ein bestimmter Startzyklus des Microprozessors 6502 (MP) initialisiert werden. Dies geschieht durch das Drücken der RS-Taste. Nun befindet sich der MP in einem wohl-definierten Teil des Monitorprogramms und die Anzeige leuchtet auf. Während dieser Einschaltphase wurden bestimmte Speicherzellen und Bereiche für bestimmte Daten reserviert u. a. m.

Will man eine bestimmte Adresse anwählen, so muß man diese nur über das Tastenfeld eingeben, denn der MP befindet sich anfangs immer im Adresszustand. Sollen Daten geändert werden, drückt man die DA-Taste und gibt sie dann über das Tastenfeld ein. Wurde die Taste DA betätigt, so bleibt der MP im Datenzustand bis er durch Drücken der Taste AD wieder in den Adresszustand versetzt wird.

Unter Zustand verstehen wir hier immer einen bestimmten Programmteil des Monitors.

Will man die Adresse inkrementieren (um 1 erhöhen), so drückt man die + Taste. Hiermit sind kurz die Speichermodifikationsmöglichkeiten des Monitors erklärt.

Man kann ein Anwenderprogramm starten, indem man die Startadresse über das Tastenfeld eintippt und dann die Taste GO drückt. Nun verläßt der MP das Monitorprogramm und die Anzeige verlöscht.

Die Fehlersuche in Anwenderprogrammen wird durch den Monitor wie folgt ermöglicht. Wird der SST-Schalter (single step) eingeschaltet, so führt der MP immer nur einen Befehl des Anwenderprogrammes aus und springt dann in den Monitor zurück. Da alle Registerinhalte in RAM-Zellen abgespeichert werden, kann man leicht durch Lesen dieser Speicherzellen den Ablauf des Programmes verfolgen oder beeinflussen.

Es lassen sich auch durch Einfügen des BRK-Befehls Unterbrechungspunkte in das Programm einfügen. Erreicht der MP diesen Punkt, so springt er in den Monitor zurück.

An Peripherie besitzt der KIM ein Tastenfeld, eine 6-stellige Siebensegmentanzeige, ein Cassettenrecorderinterface und eine Anschlußstelle für einen ASCII-Fernschreiber. Für alle diese Peripherien ist im Monitor ein entsprechendes „Bedienungsprogramm", das manchmal (z. B. Cassettenprogramm) recht aufwendig ist, vorgesehen.

2.3 Anwenderprogramme; Prinzipien der Programmierung

Unter einem Anwenderprogramm (users program) versteht man ein Programm, das eine vom Anwender gestellte Aufgabe (Problem) löst. Im Gegensatz zu einem Betriebsprogramm, das oft in Festwertspeichern (ROMs) gespeichert ist, wird ein Anwenderprogramm in den meisten Fällen in löschbare Speicher (RAMs) abgelegt, da die Anwenderprogramme häufig geändert werden. Nur bei Microcomputern, die zur Erfüllung von ganz speziellen Aufgaben (z. B. TV-Spiele) konstruiert wurden, wird das Anwenderprogramm (Spielprogramm) in ROMs gebrannt.

Bei der Entwicklung von Anwenderprogrammen hält man sich sinnvollerweise an die folgenden Prinzipien.

Die Prinzipien der Programmentwicklung:

1. Eine genaue Problemstellung
2. Die Problemanalyse
3. Entwicklung eines Lösungsverfahrens (Algorithmus)
4. Erstellung eines Ablaufplanes (Flußdiagramm)
5. Formulierung in einer Programmiersprache
6. Probeläufe mit Fehlersuche
7. Korrekturen

Die Zielsetzung dieses Buches ist es nun, diese einzelnen Schritte der Programmentwicklung anhand von vielen Beispielen einzuüben (Kap. 4). Hier wollen wir einen ersten Überblick gewinnen.

Eine genaue Problemstellung ist die Grundvoraussetzung für jegliche Programmierarbeit. Da meistens der Anwender und der Programmierer nicht die gleiche Person sind, ist eine möglichst präzise Problemstellung unumgänglich. Sehr oft wird gerade hier gesündigt. Dann erfüllt das vom Programmierer erstellte Programm nicht alle Wünsche des Anwenders, weil dieser vielleicht einige Wünsche garnicht äußerte.

Zum Beispiel könnte die Problemstellung lauten (Kap. 4.4):
Es soll eine Ampelanlage gesteuert werden. Eine so kurze Formu-

lierung ist nicht ausreichend. Es müssen noch die folgenden Fragen geklärt werden:

1. Wie sieht die Straßenkreuzung aus?
2. Wieviele verschieden geschaltete Ampeln sollen installiert werden?
3. Welche Zeitlängen sollen die verschiedenen Schaltphasen der Anlage haben?
4. Soll die Anlage leicht programmierbar (umschaltbar) auf verschiedene Schaltabläufe sein? (Tag-, Nachtbetrieb u. s. w.)

Die Problemanalyse stellt nun schon häufig eine hohe Anforderung an den Programmierer dar, der nun das Problem möglichst weitgehend zergliedern, aufteilen und gruppieren muß.

Er wird versuchen, das Problem so in Teile aufzuteilen, daß möglichst viele Teile durch schon vorhandene Programme bearbeitet werden (Unterprogrammtechnik, Struktuiertes Programmieren).

Die Entwicklung eines Lösungsverfahrens (Algorithmus) schließt sich nun an. Man hat bei der Problemanalyse vielleicht bekannte Strukturen entdeckt und kann nun den schon mal entwickelten Algorithmus für das neue Problem umformulieren. Manchmal muß man auch ganz neue Wege gehen und einen Algorithmus selber neu entwickeln. Das kann sehr zeitraubend sein und muß nicht immer zu einem Erfolg führen. Denn, was dem Mathematiker bekannt ist, nicht für jedes Problem kann ein Algorithmus gefunden werden.

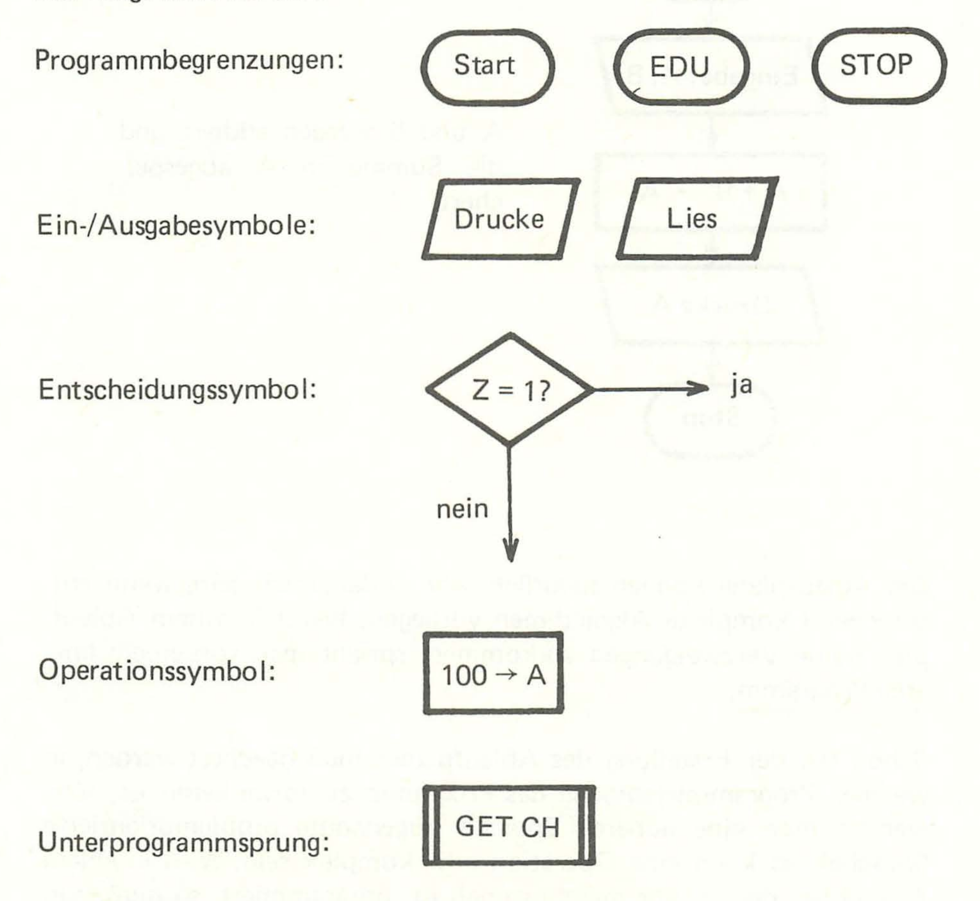
An dieser Stelle sei ausdrücklich darauf hingewiesen, daß es kein allgemeingültiges Verfahren (das man nur „lernen" muß) zum Auffinden von Algorithmen gibt.

Der Programmierer muß umfangreiches Sachwissen und große Literaturkenntnisse besitzen, um diesen Schritt der Programmentwicklung zu meistern. Für viele Probleme sind in der Literatur mehrere Algorithmen bekannt. Mit anderen Worten: Erfahrung, Literaturkenntnisse und Intuition sind die wichtigsten Werkzeuge des Programmierers, wenn er Lösungsverfahren sucht. Viele Probleme kann man in sehr kleine Einheiten auflösen, für die dann leicht Algorithmen erstellbar sind.

Ist der Algorithmus erst einmal gefunden, kann man leicht einen Ablaufplan (Flußdiagramm, flowchart) erstellen.

Manchmal sind die Algorithmen so verständlich, daß man auf den Ablaufplan verzichten kann. Oft kann man den Lösungsweg gleich in der Form eines Ablaufplanes angeben.

Ein Ablaufplan besteht aus mehreren unterschiedlichen Symbolen, die hier aufgeführt werden.

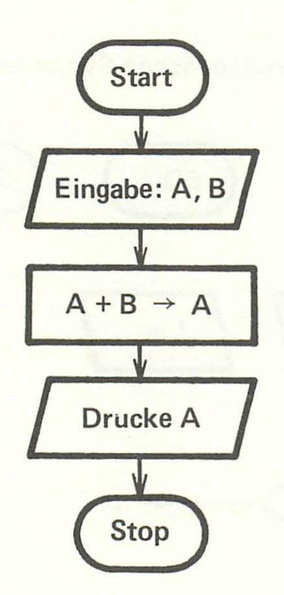


Diese wenigen Symbole reichen meistens aus. Manchmal werden noch einige Sondersymbole definiert, was wir auch an geeigneter Stelle, wenn notwendig, durchführen.

Nun ein einfaches Beispiel:

1. Das Problem: Der Computer soll zwei Zahlen addieren und auf dem Drucker die Summe ausdrucken.

2. Der Ablaufplan:



A und B werden addiert und die Summe in A abgespeichert.

Die Ablaufpläne können natürlich sehr umfangreich sein, wenn entsprechend komplexe Algorithmen vorliegen. Wenn in einem Ablaufplan keine Verzweigungen vorkommen, spricht man von einem linearen Programm.

Schon bei der Erstellung des Ablaufplanes muß beachtet werden, in welcher Programmiersprache das Programm zu formulieren ist. Verwendet man eine höhere Sprache (sogenannte problemorientierte Sprache), so kann eine Operation sehr komplex sein. Wird in einem Assembler, der ja sehr maschinennah ist, programmiert, so muß man die Operationen sehr fein wählen. Wenn bei der Erstellung des Ablaufplanes die Eigenarten der verwendeten Sprache genügend beachtet werden, ist die nun folgende Formulierung in einer Programmiersprache leicht durchführbar.

Der Programmierer muß natürlich die Syntax (Vorschriften über den Aufbau eines Programms) und die Semantik (Vorschriften über die Bedeutung der Sprachelemente) der betreffenden Programmiersprache gut beherrschen. Für jede Programmiersprache gibt es geeignete Handbücher, die man eingehend studieren sollte.

Wir wollen uns in diesem Buch auf die Programmierung im Maschinencode (Assemblerebene des 6502) beschränken. Die Syntax dieser Sprache ist sehr einfach und die Semantik wird im folgenden Kapitel (Kap. 3) vorgestellt.

Ist die Programmerstellung mit der Formulierung in einer Programmiersprache abgeschlossen, so ist die nun folgende Testphase (Probeläufe mit Fehlersuche und -korrekturen) unbedingt durchzuführen. Viele Programme selbst sehr erfahrener Programmierer laufen anfangs nicht zufriedenstellend.

Jede Phase hat ihre typischen Fehlerquellen. Das Fehlersuchen und Fehlerbeseitigen (debugging) kann mehr Zeit beanspruchen, als die erste bis fünfte Phase zusammen. Ein gutes Betriebssystem kann das Debugging wesentlich erleichtern. Auch der Aufbau (die Struktur) eines Programmes beeinflußt die Fehlersuche wesentlich. Man sollte immer ein längeres Programm in einzelne Teilprogramme zerlegen, die getrennt getestet werden.

Ist nun das Programm fehlerfrei und zufriedenstellend, sollte man die Dokumentation nicht vergessen. Jede Entwicklungsphase muß geeignet schriftlich festgehalten (dokumentiert) werden. Sehr bald kann mancher Programmierer seine eigenen Programme nicht mehr bedienen, wenn er sie schlecht dokumentierte. Nur ein gut dokumentiertes Programm kann nachträglich verbessert und abgeändert werden (mit einem vertretbaren Zeitaufwand). Die Qualität eines Programmpaketes erkennt man an der Güte der Dokumentation.

3. Der 6502

Nun soll der Microprozessor 6502 vorgestellt werden.
Erst werfen wir einen Blick auf die Hardware (innere Struktur des Chips) um uns dann seiner Programmierung zuzuwenden. Wir lernen hier den Assembler (Maschinencode, Maschinensprache) des 6502 kennen. Die Syntax (der Sprachaufbau) ist so einfach, daß man sie schnell mit Hilfe einiger Beispiele kennenlernt. Die Semantik (Bedeutung der Sprachelemente) zu lernen, ist ein umfangreiches Vorhaben, bei dem dieses Buch behilflich sein will.

Sicherlich wird mancher Leser mit der Kürze dieses Kapitels nicht einverstanden sein. Ein nachträgliches Studium eines Programmierhandbuches eines Herstellers des 6502 ist gewiß für den einen oder anderen Leser notwendig. Hier soll auch nur das allernotwendigste Rüstzeug zum Verständnis der Beispiele von Kapt. 4 gebracht werden.

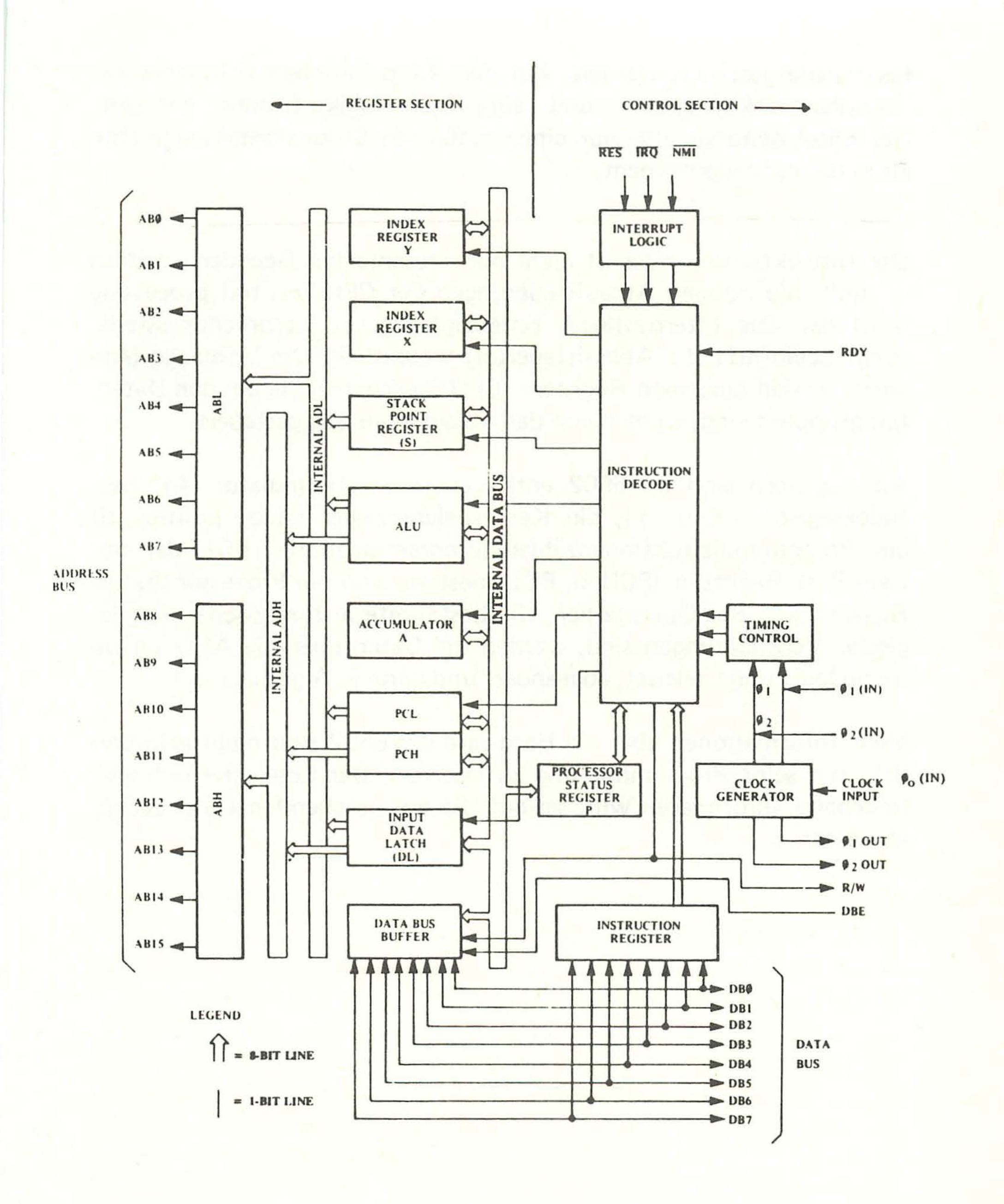
Schlagen Sie, wenn sich beim Studium der Beispiele von Kap. 4 Fragen ergeben, immer wieder zu Kap. 3 zurück und lesen Sie die entsprechenden Absätze noch einmal durch. Dann werden Sie sicherlich auch nach einiger Übung den Befehlssatz des 6502 verstehen.

3.1 Die innere Struktur

Abb. 3.1 zeigt uns die wesentlichen Elemente des Microprozessors 6502. Wie man leicht erkennen kann, erinnert die Struktur an die Busstruktur eines Computers. Im Inneren des Chips wird also ebenfalls an der Busstruktur festgehalten.

Der innere Datenbus wird über bidirektionale Treiber (databus buffer), die vom Instruktionsdecoder (instruction decode) kontrolliert werden, an den Systembus angeschlossen. Der innere Adressbus steuert über die Treiber ABL und ABH den Systemadressbus. Das Instruktionsregister (instruction register) ist direkt an den Systembus angekoppelt. In ihm wird das Befehlsbyte (instruction) gespeichert. Dieses Befehlsbyte wird im Instruktionsdecodierer entschlüsselt (decodiert).

Die recht komplizierten inneren Abläufe müssen durch Zweiphasen-



1. Der Clock-Generator ist im R6512 bis R6515 nicht enthalten.
2. Die Adressierungsmöglichkeit und Kontrollmöglichkeit ist bei jedem der R650X-Produkte unterschiedlich.

taktsignale gesteuert werden. Auf dem Chip befinden sich daher ein Zweiphasentaktgenerator und eine Steuerlogik (timing control). Der 6502 benötigt also nur einen einfachen Einphasentaktgenerator als externes Steuerelement.

Der Instruktionsdecoder ist nicht nur ein einfacher Decoder, sondern er stellt die nötigen Ablaufsteuerungen der CPU (central processing unit) dar. Die Interruptlogik (interrupt logic, Unterbrechungssteuerung) beeinflußt die Ablaufsteuerung wesentlich. Die Vorgänge (Abläufe) in den einzelnen Registern (CPU-Speichern), die um den Datenbus gruppiert sind, werden von der Ablaufsteuerung gesteuert.

An Registern sind im 6502 enthalten: ein Akkumulator (A), zwei Indexregister (X u. Y), ein Kellerspeicherzeiger (stack pointer, S) ein Programminstruktionenzähler (programmcounter, PC), der aus zwei 8bit Registern (PCH u. PCL) besteht, und ein Prozessor-Status-Register (P). Bei Operationen, die bestimmte arithmetische oder logische Verknüpfungen sind, werden die Daten über die ALU (arithmetic logic unit) geleitet, abgeändert und dann in A gespeichert.

Mehr Informationen über die Hardware des 6502 sind nicht erforderlich, um seine Programmierung zu erlernen. Der Leser, der sich weitergehend informieren will, sei auf die entsprechenden Datenbücher verwiesen.

3.2 Die Programmierung des 6502

3.2.1 Das Programmiermodell

Die sehr komplexe innere Struktur des 6502 ist für den Programmierer nicht von wesentlicher Bedeutung. Ihm reicht ein einfaches Programmiermodell der CPU, das er sich anfertigt. In Abb. 3.2.1 ist ein solches Modell abgebildet.

Hier sind nur die einzelnen Register angedeutet, die der Programmierer direkt oder indirekt durch Befehle beeinflussen kann. Wir wollen anhand dieses Modells die Funktionen der einzelnen Register kennenlernen.

Der Akkumulator, ein 8 bit Register, ist das wichtigste Datenregister der CPU. In ihm werden die Ergebnisse aller arithmetischer und logischer Operationen abgespeichert. Er liefert vor den Operationen immer den ersten Operanden; der zweite, wenn notwendig, wird aus irgendeiner Speicherstelle entnommen.

Die Indexregister X und Y sind einmal als Zwischenspeicher für Teilergebnisse einsetzbar, es lassen sich ein paar arithmetische Operationen mit ihnen durchführen und das X-Register kann den Stackpointer laden beziehungsweise lesen. Eine weitere, wichtige Eigenschaft beider Indexregister liegt aber in ihrer Verwendung bei der indizierten und der indirekten Adressierung (s. Kap. 3.2.3).

Die Vorgänge im Akkumulator oder in den Indexregistern beeinflussen die einzelnen Bits des Prozessor-Status-Registers (P). Dieses Register ist als eine Vereinigung einzelner Bits (flags) zu verstehen. Bei den sogenannten bedingten Programmverzweigungen ist der Inhalt eines Flags für den weiteren Ablauf des Programms bestimmend.

Der Programmzähler oder Programmzeiger (PC) ist das einzige 16 bit Register des 6502. In ihm ist die Adresse immer des folgenden Befehlsbytes abgespeichert. Er steuert, allgemein formuliert, den Zugriff zum Speicher. Wird der Programmzähler nicht beeinflußt, so zählt er nach jedem Speicherzugriff automatisch um 1 höher. Man kann den Inhalt des Programmzählers allerdings beeinflussen, durch einen Sprungbefehl zum Beispiel.

Nicht alle Microprozessoren besitzen einen automatischen Kellerspeicherzeiger (stackpointer). Unter einem Kellerspeicher (stack) versteht man einen Bereich im Datenspeicher, der wie ein first-in-last-out-Register strukturiert ist. Der Stackpointer zeigt immer automatisch die aktuelle Adresse im Stack an. Nach einem Schreibbefehl erniedrigt sich sein Inhalt automatisch um 1 (decrementiert) und nach einem Lesebefehl erhöht er sich um 1 (incrementiert). Es gibt spezielle Lese- und Schreibbefehle für den Kellerspeicher (siehe Kap. 3.3). Da der Zeiger des 6502 8 bit umfaßt, kann der Kellerspeicher maximal 256 bytes beinhalten. Der Zeiger ist durch die Hardware auf die 1. Seite des Speichers (RAM) 0100 bis 01FF festgelegt.

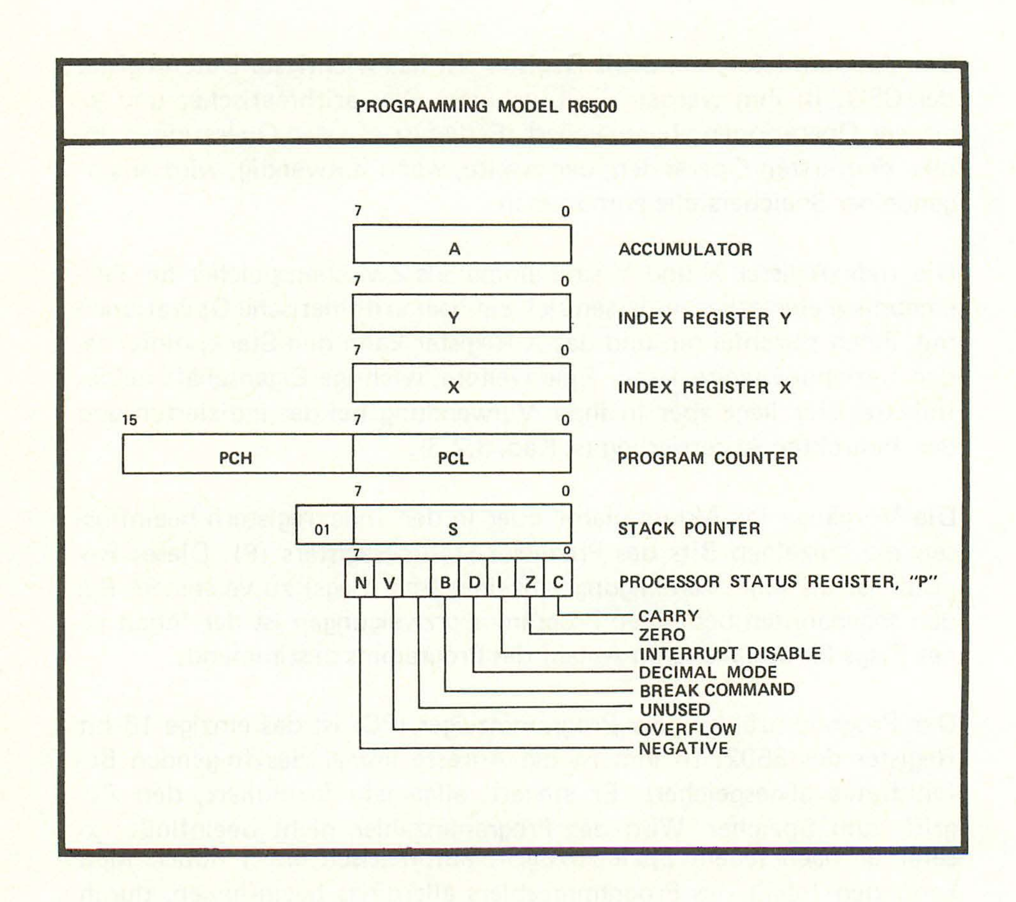
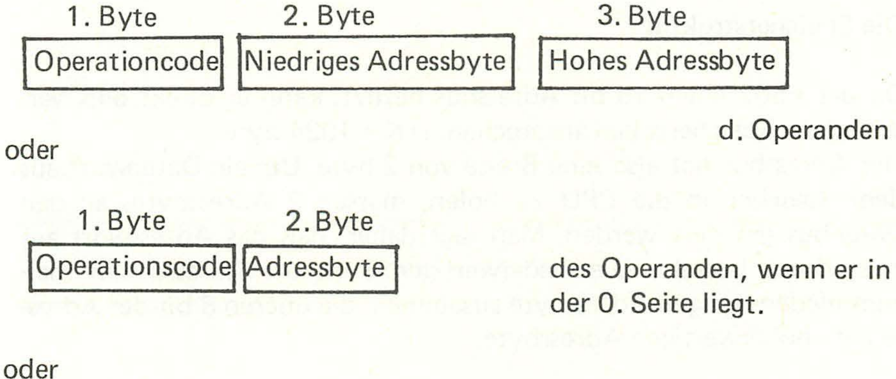


Abb. 3.2.1

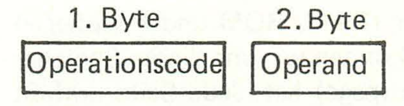
3.2.2 Die Befehlsstruktur

Bei einer Wortlänge von 8 bit oder weniger, muß ein Befehl im allgemeinen mehr als ein Wort (byte) beanspruchen.
Der 6502 besitzt 3 byte, 2 byte oder 1 byte Befehle.

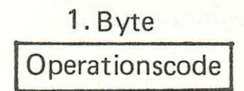
Ein Befehl hat eine der folgenden Strukturen:



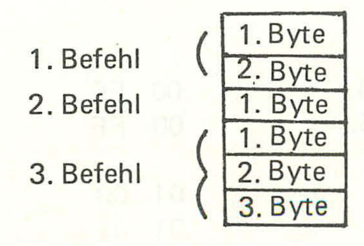
oder



oder



Die einzelnen Bytes der Befehle werden der Reihe nach im Programmspeicher abgelegt.
Ein Ausschnitt aus einem Programm könnte so aussehen:



3.2.3 Die Adressierungsmethoden

Der 6502 bietet eine Vielzahl von verschiedenen Adressierungsmethoden an, die die einzelnen Befehle sehr flexibel gestalten.

Unter Adressierung wollen wir die Methoden verstehen, mit denen auf das Datenbyte (Operand) zugegriffen wird.

Die Speicherstruktur

Da der 6502 einen 16 bit Adressbus besitzt, kann er direkt 64K verschiedene Speicherzellen ansprechen. (1K = 1024 byte)
Der Adressbus hat also eine Breite von 2 byte. Um ein Datenwort aus dem Speicher in die CPU zu holen, müssen 2 Adressbytes an den Datenbus geliefert werden. Man sagt daher, daß das Adresswort aus zwei Bytes besteht. Die niederwertigen 8 bit der Adresse faßt man zum niederwertigen Adressbyte zusammen, die oberen 8 bit der Adresse zum höherwertigen Adressbyte.

Diese Einteilung des Adresswortes läßt sich auch sehr anschaulich erklären. Nehmen wir an, unser Speicher (RAM, ROM und Peripherie) umfasse 64K byte (= 65536 byte), so können wir uns diesen Speicher als ein Buch vorstellen, das 256 Seiten (page) hat. Jede Seite umfaßt dann 256 Worte. Das hohe Adressbyte kennzeichnet dann die Seiten und das niedrige Adressbyte kennzeichnet das betreffende Wort innerhalt der Seite. Die Wort- und Seitennummerierung beginnt mit Null.

	dezimal Wort	hexal Wort
	0.	00 00
	1.	00 01
	2.	00 02
0. Seite	.	.
	.	.
	254.	00 FE
	255.	00 FF
	0.	01 00
	1.	01 01

1. Seite

:   :

255.   01 FF

1. Die implizierte (implied) Adressierung

Hierunter werden alle 1 byte Befehle zusammengefaßt. Bei dieser Adressierung ist kein Datenbyte und kein Adressbyte notwendig.

2. Die unmittelbare (immediate) Adressierung

Befehle mit dieser Adressierung sind 2 byte Befehle. Das dem Operationsbyte folgende Byte beinhaltet den Operanden (Datenbyte). Diese Adressierung benutzt man z. B. zum Laden von CPU-Registern mit einer Konstanten.

3. Die absolute, direkte (absolute) Adressierung

Die Befehlslänge ist nun 3 byte. Die beiden, dem Operationsbyte folgenden Bytes beinhalten die vollständige Adresse des Operanden.

4. Die nullseitige (zeropage) Adressierung

Dies ist eine typische Eigenart des 6502. Die Befehlslänge beträgt 2 byte. Das dem Operationsbyte folgende Byte beinhaltet den niederwertigen (low order) Teil des Adressbytes des Operanden. Der höherwertige (high order) Teil wird auf 00 gesetzt. Man kann also Operanden nur in der 0. Seite (Adresse 00 00 bis 00 FF) adressieren. Die nullseitige Adressierung hat gegenüber der absoluten zwei Vorzüge:

1. Die Ausführungszeiten der Befehle werden verkürzt,
2. Die Länge des Programmes wird kürzer.

5. Die Akkumulator Adressierung

Einige Befehle verändern nur den Akkumulatorinhalt, sie sind 1 byte Befehle.

6. Die relative (relative) Adressierung

Auch diese Adressierung ist eine typische Eigenart des 6502. Sie ist auf die bedingten Sprungbefehle beschränkt. Die Länge eines Befehls beträgt zwei Bytes.

An einem Beispiel wollen wir uns diese Adressierung erklären. Nehmen wir an, die CPU stößt auf den Befehl BCC (springe, wenn C = 0) und C sei gelöscht, so muß die CPU nun das Sprungziel erfahren. Es wäre möglich, in den beiden folgenden Bytes die Adresse des Sprungziels zu laden (absolute Adressierung).
Bei der relativen Adressierung geht man nun anders vor. Es wird die Anzahl der Bytes, die übersprungen werden sollen, bestimmt und in das dem Befehlsbyte folgende Byte geladen. Nun muß die CPU noch zwischen Vorwärts- oder Rückwärtsspringen unterscheiden können. Dies geschieht dadurch, daß man bei Rückwärtssprüngen in das 2. Byte das 2er-Komplement (neg. Zahl) der Sprungzahl eintragen muß.

```
1. Vorwärtssprung:  BCC R M1
                         57 Bytes
                    M1 ........
```

Es sollen also 57 Bytes übersprungen werden.

$57 = 00111001_2 = 39_{hex}$

Wir tragen in den Speicher (RAM) folgenden Befehl ein:

```
BCC 90    1. Byte
    39    2. Byte
```

```
2. Rückwärtssprung:  M1  LDA  AB  0015
                                  52 Bytes
                         BCC R M1
```

Es sollen 57 Bytes zurück übersprungen werden. Wir müssen die 2 Bytes des Befehls BCC und die 3 Bytes des Befehls LDA mit hinzunehmen!

$57 = 00111001_2$ $\quad$ $57 = 00111001_2$

$$
\begin{array}{rl}
57 = 00111001_2 & \\
11000110 & \text{komplementär} \\
+1 & \\
\hline
11000111 & \text{2er - Komplement} \\
= C7_{hex} &
\end{array}
$$

Wir tragen nun in den Speicher den folgenden Befehl ein:

BCC 90	1. Byte
C7	2. Byte

Die relative Adressierung hat einen gravierenden Vorteil. Sie vereinfacht die Verlagerung von Programmen im RAM wesentlich. Die Nachteile sollen nicht verschwiegen werden. Man kann nur maximal 127 Bytes vorwärts und 128 Bytes rückwärts überspringen. In den seltenen Fällen, in denen größere Sprungweiten erforderlich sind, müssen an geeigneten Stellen unbedingte Sprungbefehle (JMP) eingebaut werden. Die Berechnung der relativen Adressen ist etwas mühsam. Man kann sich ein Hilfsprogramm schreiben, das diese Arbeit übernimmt (siehe Kap. 4.1.1.).

7. Die indirekte (indirect) Adressierung

Bei dieser Adressierungsmethode beinhalten die dem Operationsbyte folgenden 2 Bytes nicht die aktuelle Adresse, sondern die Adresse der Speicherstelle im Speicher (ROM oder RAM), in der das niederwertige Adressbyte des Operanden abgespeichert ist. In der folgenden Speicherzelle befindet sich dann das höherwertige Byte der Adresse des Operanden.

Nur ein Befehl des 6502 besitzt diese reine indirekte Adressierung, das ist der Befehl JMP. Diesen Befehl kann man daher nutzen, um den Programmzähler mit einem neuen Adressbyte zu laden. Er ist besonders bei Programmverzweigungen mit vielen möglichen Wegen mit Vorteil zu benutzen.

Als nächstes wollen wir uns den sechs verschiedenen Indexregister-Adressierungen zuwenden. Diese, zum Teil sehr mächtigen Methoden,

sind eine weitere besondere Stärke des 6502. Wie in Kapitel 3.2.1 schon erwähnt wurde, haben die beiden Indexregister (X und Y) bei der Adressierung von Operanden während einer Befehlsausführung eine besondere Aufgabe. Mit ihrer Hilfe kann man nämlich Speicherzellen „indizieren". Das heißt, Daten mit einem Index versehen. Bei Tabellenverarbeitungen ist dies notwendig. Die Verwendung der verschiedenen Indizierungsmethoden wird an geeigneter Stelle eingeübt (Kap. 4). Hier seien sie nur kurz vorgestellt.

8. Die absolut durch X indizierte Adressierung (absolute indexed with X)

Diese Adressierungsmethode ist leicht verständlich. Der Inhalt des Registers X wird nämlich zu den dem Operationsbyte folgenden zwei Adressbytes addiert, um die aktuelle Adresse des Operanden zu bestimmen.

Ein Beispiel: X habe den Inhalt 05,
der Befehl sei BD 07 02.

Es wird die Addition $02\ 07_{hex}$

$$\begin{array}{r} 02\ 07_{hex} \\ 05 \\ \hline = 02\ 0C_{hex} \end{array}$$

durchgeführt

Der aktuelle Operand hat die Adresse $02\ 0C_{hex}$.

9. Die absolut durch Y indizierte Adressierung (absolute with Y indexed)

Diese Adressierungsmethode ist die gleiche wie oben, nur wird jetzt der Inhalt des Y-Registers verwendet.

10. Die nullseitige durch X indizierte Adressierung (zeropage indexed with X)

Diese Adressierungsmethode unterscheidet sich nicht wesentlich von der absolut indizierten. Die Befehle umfassen nun aber nur zwei Bytes und die Adressen beschränken sich auf die 0. Seite. Die Vorteile sind eine Verkürzung der Ausführungszeiten und eine Verringerung des Programmumfangs.

11. Die nullseitige durch Y indizierte Adressierung (zeropage indexed with Y)

Es gilt hier sinngemäß das Gleiche wie unter 10., nur ist nun der Inhalt des Y-Registers entscheidend.

Nun folgen die beiden mächtigsten Adressierungsmethoden, die den Programmaufwand bei der Verarbeitung von Tabellen und Listen ganz erheblich reduzieren. Der Programmieranfänger kann diese beiden Verfahren ruhig vorerst übergehen. Sie werden nur in komplexeren Programmen verwendet.

12. Die durch X indizierte indirekte Adressierung (indexed indirect adressing)

Wir betrachten Abb. 3.2.12. Wir nehmen an, daß irgendwo im Speicher Daten (DATA1, DATA2,...) gespeichert sind, die wir mit einem

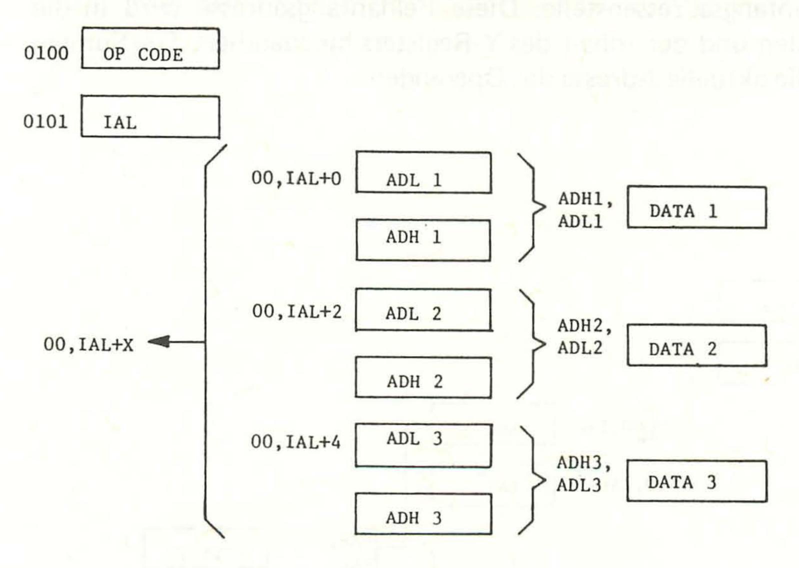


Abb. 3.2.12

Index versehen. Die Adressen dieser Daten sollen in der Reihenfolge der Indizes in der 0. Seite angeordnet sein. Die Adresse (2 Bytes) des 1. Datenbytes (DATA1) zuerst u.s.w.. Der Zugriff zu den Daten läuft

nun wie folgt ab. Das dem Operationsbyte folgende Byte muß die Adresse IAL des 1. Adressbytes beinhalten. Der Inhalt des X-Registers wird nun dazuaddiert (er muß geradzahlig sein!). Diese Summe weist nun auf das aktuelle Adressbytepaar hin. Diese wird in die CPU geladen und nun das Datenbyte, dessen absolute Adresse geladen wurde, adressiert.

13. Die indirekte durch Y indizierte Adressierung (indirect indexed adressing)

Wir betrachten jetzt die Abbildung 3.2.12. Wir nehmen an, daß sich irgendwo im Speicher ein angeordnetes Datenfeld (DATA1, DATA2...) befindet. Durch die Anordnung sind die Daten mit einem Index versehen. Die erste Adresse des Datenfeldes ist in der 0. Seite gespeichert. Den Zugriff zu einem bestimmten Datenbyte im Datenfeld gestaltet die CPU wie folgt. Der Index wird in das Register Y geladen. Das dem Operationsbyte folgende Adressbyte beinhaltet die Nullseitenadresse der Feldanfangsadressenstelle. Diese Feldanfangsadresse wird in die CPU geladen und der Inhalt des Y-Registers hinzuaddiert. Die Summe ist dann die aktuelle Adresse des Operanden.

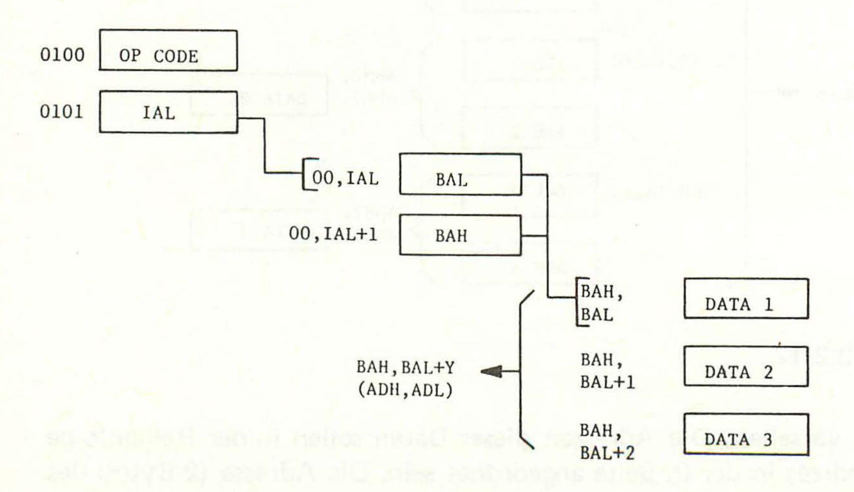


Abb. 3.2.13

3.3 Der Befehlssatz

Nun wollen wir den Befehlssatz des 6502 kennenlernen.
Hier wird der Operationscode (Hexcode) eines jeden Befehls, seine genaue Funktion und seine Darstellung im Assemblercode (Mnemonic) vorgestellt. Dieses Kapitel ist in zwei Teile gegliedert. Im ersten Teil werden die Befehle, in Gruppen eingeteilt, vorgestellt. Der zweite Teil besteht aus Tabellen, die dem Leser die Übersicht erleichtern sollen. Sicherlich wird man häufig beim Lesen des 4. Kapitels hier nachlesen müssen.

3.3.1 Die Beschreibung der Befehle

Der eigentlichen Beschreibung der Funktion eines Befehls folgt eine Kurzdarstellung der Operation, der Flagbeeinflussung, des Assemblercodes und eine Angabe der Operationscode bei den verschiedenen Adressierungen. Für die Adressierungen werden die folgenden Abkürzungen verwendet:

		Mnemonic
implied		IMP
immediate		IM
absolute		AB
zeropage		ZP
accumulator		AC
relative		R
indirect		IND
absolute indexed	with X	ABX
	with Y	ABY
zeropage indexed	with X	ZPX
	with Y	ZPY
indexed indirect		INX
indirect indexed		INY

1. Gruppe: Transportbefehle

Unter Transportbefehlen versteht man Befehle, die einen Datenaustausch zwischen Peripherie und CPU ermöglichen. Es gibt grundsätzlich drei Transportrichtungen:

1. Richtung: CPU → Peripherie,
2. Richtung: Peripherie → CPU
3. Richtung: CPU → CPU

a) CPU → Peripherie

STA: Speichere den Akkumulatorinhalt in eine Speicherzelle ab. Mit diesem Befehl kann man natürlich auch einen Ausgabepuffer laden.

kurz: A → M Flags: keine

STA	AB 8D
	ZP 85
	INX 81
	INY 91
	ZPX 95
	ABX 9D
	ABY 99

STX: Speichere den Inhalt des X-Registers in eine Speicherzelle oder einen Ausgabepuffer ab.

kurz: X → M Flags: keine

STX	AB 8E
	ZP 86
	ZPY 96

STY: Speichere den Inhalt des Y-Registers in eine Speicherzelle oder einen Ausgabepuffer ab.

kurz: Y → M Flags: keine

STY	AB 8C
	ZP 84
	ZPX 94

b) Peripherie → CPU

Bei den drei folgenden Befehlen werden das N- und das Z-Flag des P-Registers wie folgt beeinflußt.

Ist das Bit 7 des Datenbytes 1, so wird N = 1, andernfalls 0.
Sind alle Bits des Datenbytes 0, so wird Z = 1, andernfalls 0.

LDA: Lade den Inhalt einer Speicherzelle oder eines Eingabepuffers in den Akkumulator.

			IM A9
kurz: M → A	Flags: N Z	LDA	AB AD
			ZP A5
			INX A1
			INY B1
			ZPX B5
			ABX BD
			ABY B9

LDX: Lade den Inhalt einer Speicherzelle oder eines Eingabepuffers in das X-Register.

kurz: M → X	Flags: N Z	LDX	IM A2
			AB AE
			ZP A6
			ABY BE
			ZPY B6

LDY: Lade den Inhalt einer Speicherzelle oder eines Eingabepuffers in das Y-Register.

kurz: M → Y	Flags: N Z	LDY	IM A0
			AB AC
			ZP A4
			ABX BC
			ZPX B4

c) Transporte innerhalb der CPU

Bei den hierzu gehörigen Befehlen werden auch das Z- und N-Flag in der gleichen Art wie bei b) beeinflußt. Nur beim Befehl TXS findet keine Flagbeeinflussung statt.

TAX: Bringe den Inhalt des Akkumulators in das X-Register.

kurz: A → X	Flags: N Z	TAX	IMP AA

TAY: Bringe den Inhalt des Akkumulators in das Y-Register.

kurz: A → Y	Flags: N Z	TAY	IMP A8

TSX: Bringe den Inhalt des Stackzeigers in das X-Register.

kurz: S → X	Flags: N Z	TSX	IMP BA

TXA: Bringe den Inhalt des X-Registers in den Akkumulator.
kurz: X → A   Flags: N Z   TXA   IMP 8A

TXS: Bringe den Inhalt des X-Registers in den Stapelzeiger.
kurz: X → S   Flags: keine   TXS   IMP 9A

TYA: Bringe den Inhalt des Y-Registers in den Akkumulator.
kurz: Y → A   Flags: N Z   TYA   IMP 98

2. Die arithmetischen Operationen

Unter arithmetischen Operationen verstehen wir Operationen, die Rechenabläufe erzeugen.

Alle arithmetischen Operationen beeinflussen das N- und das Z-Flag. Die Befehle ADC, CMP, CPX, CPY und SBC beeinflussen zusätzlich noch das C-Flag. Nur die Befehle ADC und SBC beeinflussen auch das V-Flag. Das C-Flag zeigt eventuell entstehende Überträge an. Das V-Flag wird bei vorzeichenbehafteter Arithmetik zur Bestimmung von Überläufen herangezogen. Es entspricht dem 7. Bit des Rechenergebnisses. Wir wollen uns hier nicht weiter mit dem V-Flag befassen, da in diesem Buch keine Rechenprogramme mit Vorzeichenarithmetik erstellt werden.
Eine typische Eigenart des 6502 ist die Möglichkeit, dezimal (im BCD-Code) zu rechnen. Durch Setzen des D-Flags kann die CPU in den dezimalen Mode geschaltet werden. Die Befehle ADC und SBC arbeiten dann dezimal, die anderen Befehle verbleiben im dualen Mode.

ADC: Das C-Flag und das adressierte Byte werden zum Inhalt des Akkumulators addiert. Die Summe wird in den Akkumulator gebracht. Wenn ein Überlauf entstand, wird das C-Flag gesetzt, andernfalls wird es rückgesetzt. Der Programmierer muß vor einer Addition das C-Flag löschen, da sonst ein falsches Ergebnis entstehen kann.
kurz: A + M + C → A
Flags: N Z C V   ADC   IM 69
AB 6D
ZP 65   ABX 7D
INX 61   ABY 79
INY 71
ZPX 75

CMP: Das adressierte Byte wird vom Inhalt des Akkumulators subtrahiert. Der Akkumulatorinhalt wird aber nicht verändert. Dieser Befehl beeinflußt also nur die Flags.
kurz: A − M Flags: N Z C CMP IM C9
AB CD
ZP C5
INX C1
INY D1
ZPY D5
ABX DD
ABY D9

CPX: Das adressierte Byte wird vom Inhalt des X-Registers subtrahiert. Der Inhalt von X bleibt aber unverändert.
Dieser Befehl beeinflußt nur die Flags.
kurz: X−M Flags: N Z C CPX IM E0
AB EC
ZP E4

CPY: Das adressierte Byte wird vom Inhalt des Y-Registers subtrahiert. Der Inhalt von Y bleibt unverändert.
Dieser Befehl beeinflußt nur die Flags.
kurz: Y−M Flags: N Z C CPY IM C0
AB CC
ZP C4

DEC: Der Inhalt der adressierten Speicherzelle wird um 1 erniedrigt (decrementiert).
kurz: M−1 → M Flags: N Z DEC AB CE
ZP C6
ZPX D6
ABX DE

DEX: Das X-Register wird decrementiert.
kurz: X−1 → X Flags: N Z DEX IMP CA

DEY: Das Y-Register wird decrementiert.
kurz: Y−1 → Y Flags: N Z DEY IMP 88

INC: Der Inhalt der adressierten Speicherzelle wird um 1 erhöht (incrementiert).
kurz: M+1→M Flags: N Z INC AB EE
ZP E6
ZPX F6
ABX FE

INX: Das X-Register wird incrementiert.
kurz: X+1→X Flags: N Z INX IMP E8

INY: Das Y-Register wird incrementiert.
kurz: Y+1→Y Flags: N Z INY IMP C8

SBC: Der Inhalt der adressierten Speicherstelle wird vom Akkumulatorinhalt subtrahiert. Ist C=0, so wird zusätzlich decrementiert. Muß eine 1 geborgt werden (Bereichsüberschreitung), so wird C=0, andernfalls wird C=1. Der Programmierer muß also vor der ersten Subtraktion das C-Flag setzen! Die Differenz steht im Akkumulator.
kurz: $A-M-\overline{C} \rightarrow A$
Flags: N Z C V SBC IM E9
AB ED
ZP E5
INX E1
INY F1
ZPX F5
ABX FD
ABY F9

3. Die logischen Operationen

AND: Es wird eine bitweise Undverknüpfung zwischen dem adressierten Byte und dem Akkumulatorinhalt durchgeführt. Die Verknüpfungsergebnisse werden stellenrichtig in den Akkumulator abgelegt.
Die Und-Verknüpfung für ein Bit:

∧	0	1
0	0	0
1	0	1

kurz: A ∧ M → A Flags: N Z AND

IM 29	INY 31
AB 2D	ZPX 35
ZP 25	ABX 3D
INX 21	ABY 39

EOR: Es wird eine Exclusivoder-Verknüpfung bitweise zwischen dem adressierten Byte und dem Akkumulatorinhalt ausgeführt. Die Ergebnisse werden stellenrichtig in den Akkumulator abgelegt.

Die Exclusivoder-Verknüpfung eines Bit:

∀	0	1
0	0	1
1	1	0

kurz: A∀M→A Flags: N Z EOR

IM 49
AB 4D
ZP 45
INX 41
INY 51
ZPX 55
ABX 5D
ABY 59

ORA: Es wird eine Oder-Verknüpfung bitweise zwischen dem adressierten Byte und dem Akkumulatorinhalt ausgeführt. Die Ergebnisse werden stellenrichtig in den Akkumulator abgelegt.
Die Oder-Verknüpfung für ein Bit:

V	0	1
0	0	1
1	1	1

kurz: A ∨ M → A Flags:N Z ORA

IM 09
AB 0D
ZP 05
INX 01
INY 11
ZPX 15
ABY 1D
ABY 19

4. Die Verschiebebefehle

ASL: Das adressierte Byte wird um ein Bit nach links verschoben. Das 7. Bit kommt in das C-Flag und in das 0. Bit wird eine 0 nachgeschoben.

kurz: C ← [7 0] ← 0	ASL	AB 0E	ZPX 16
Flags: N Z C		ZP 06	ABX 1E
		AC 0A	

BIT: Dieser Befehl ist eine Besonderheit des 6502. Er verändert den Akkumulatorinhalt nicht, beeinflußt also nur die Flags. Es findet zwischen der adressierten Speicherzelle und dem Akkumulator eine bitweise UND-Verknüpfung statt. Ist das Ergebnis für alle Bits 0, so wird das Z-Flag gesetzt, andernfalls wird es gelöscht. Das N-Flag wird gleich dem 7. Bit der adressierten Speicherzelle und das V-Flag gleich dem 6. Bit gesetzt.

BIT AB 2C
ZP 24

LSR: Das adressierte Byte wird um ein Bit nach rechts verschoben. Das 0. Bit kommt in das C-Flag und in das 7. Bit wird eine 0 nachgeschoben.

kurz: 0→ [7 0] → C

LSR AB 4E
ZP 46
AC 4A
ZPX 56
ABX 5E

ROL: Das adressierte Byte wird um ein Bit zyklisch nach links verschoben. Das 7. Bit kommt ins C-Flag, und das C-Flag wird in das 0. Bit geschoben.

kurz: [7 0] ← C (C-Flag ← Bit 7)

ROL AB 2E
ZP 26
AC 2A
ZPX 36
ABX 3E

ROR: Das adressierte Byte wird um ein Bit zyklisch nach rechts verschoben. Das 0. Bit kommt in das C-Flag, und das C-Flag wird an das 7. Bit geschoben.

kurz: →C→[7 0]→ (zurück zu C)

Flags: N Z C

ROR AB 6E
ZP 66
AC 6A
ZPX 76
ABX 7E

5. Die Beeinflussung der Flags

Damit der Programmierer die Flags unter Kontrolle halten kann, muß er die Zustandsbits des P-Registers einzeln setzen und löschen können.

CLC: Löscht das C-Flag.
kurz: 0 → C CLC IMP 18

CLD: Löscht das D-Flag. Es wird wieder auf den dualen Mode umgeschaltet.
kurz: 0 → D CLD IMP D8

CLI: Das Interrupt-Flag wird gelöscht. Nun ist die CPU bereit, Unterbrechungen auszuführen.
kurz: 0 → I CLI IMP 58

CLV: Das V-Flag wird gelöscht.
kurz: 0 → V CJV IMP B8

SEC: Das C-Flag wird gesetzt.
kurz: 1 → C SEC IMP 38

SED: Das D-Flag wird gesetzt. Die CPU wird auf dezimale Betriebsart umgeschaltet.
kurz: 1 → D SED IMP F8

SEI: Das I-Flag wird gesetzt. Die CPU kann nun keine Unterbrechung bedienen.
kurz: 1 → I SEI IMP 78

6. Die Sprungbefehle

a) Der unbedingte Sprung

JMP: Springe an die spezifizierte Programmstelle.
JMP AB 4C
IND 6C

b) Die bedingten Sprungbefehle

Alle bedingten Sprungbefehle werden relativ adressiert.

BCC:	Springe, wenn C = 0.	BCC	R	90
BCS:	Springe, wennC = 1.	BCS	R	B0
BEQ:	Springe, wenn Z = 1.	BEQ	R	F0
BMI:	Springe, wenn N = 1	BMI	R	30
BNE:	Springe, wenn Z = 0.	BNE	R	D0
BPL:	Springe, wenn N = 0.	BPL	R	10
BVC:	Springe, wenn V = 0.	BVC	R	50
BVS:	Springe, wenn V = 1.	BVS	R	70

7. Die Stackoperationen

Zur Bedienung des Kellerspeichers stehen dem Programmierer 4 Befehle zur Verfügung. Der Zeiger wird durch diese Befehle beeinflußt. Lesebefehle incrementieren ihn und Schreibbefehle decrementieren ihn.

PHA: Lege den Akkumulatorinhalt in den Kellerspeicher ab.
PHA IMP 48

PHP: Lege den Inhalt des P-Registers in den Kellerspeicher ab.
PHP IMP 08

PLA: Hole das unterste Datenbyte des Kellerspeichers in den Akkumulator.
PLA IMP 68

PLP: Hole das unterste Datenbyte des Kellerspeichers in das P-Register.
PLP IMP 28

8. Die Unterprogrammbehandlung

In Kapitel 4.1.1 (8. Beispiel) wird die Unterprogrammtechnik vorgestellt.
Der 6502 besitzt zwei Befehle, die die Programmierung von Unterprogrammsprüngen wesentlich erleichtern.

JSR: Springe an die angegebene Adresse und lege die, diesem Befehl folgende Adresse (Rücksprungadresse) in den Stack ab.
JSR AB 20

RTS: Der letzte Befehl eines Unterprogramms.
Kehre aus dem Unterprogramm in das Hauptprogramm zurück, indem die Rücksprungadresse aus dem Stack in den Programmzähler geladen wird.
RTS IMP 60

9. Reset- und Interruptbehandlung

Die Adressvektoren

Durch die verschiedenen Sprungbefehle wird der Inhalt des Programmzählers beeinflußt. Die dem Operationsbyte folgenden Adressbytes beinhalten den neuen Inhalt des Programmzählers oder werden zum alten Inhalt hinzuaddiert (relative Adressierung).

Der Inhalt des Programmzählers kann aber auch durch externe Signale, die an die entsprechenden Eingänge der CPU anliegen, beeinflußt werden. Drei externe Signale kann der 6502 annehmen, das RST-, das NMI- und das IRQ-Signal.

Wird einer dieser Signaleingänge aktiviert, holt sich die CPU die Sprungadresse aus einem der sogenannten Adressvektoren. Die obersten Zellen des Speicherbereiches sind als Adressvektoren ausgezeichnet.
Die Adressvektoren befinden sich in den folgenden Speicherzellen.

NMI-Impuls:	NMIL	FF	FA
	NMIH	FF	FB
RST-Impuls:	RSTL	FF	FC
	RSTH	FF	FD
IRQ-Impuls:	IRQL	FF	FE
	IRQH	FF	FF

Wird zum Beispiel der RST-Eingang der CPU aktiviert, läd sie den Programmzähler mit dem Inhalt der Zellen FF FC und FF FD. Das bedeutet einen Sprung an die durch den Adressvektor bestimmte Speicherstelle.

Die Reset- oder Restartoperation

Wird die CPU eingeschaltet, befindet sie sich in einem nicht definierbaren Zustand. Der Programmierer muß dafür sorgen, daß der $\overline{RES}$-Eingang aktiviert wird. Sobald der $\overline{RES}$-Eingang aktiviert wird, holt sich die CPU den Resetvektor aus den Zellen FF FC und FF FD und läd ihn in den Programmzähler. Wurde in den Vektor zum Beispiel die Anfangsadresse des Systemmonitors geladen, beginnt die CPU mit der Abarbeitung des Monitorprogramms.

Trat während eines Testlaufes an irgendeiner Programmstelle ein Fehler auf, der die CPU in einen unkontrollierbaren Zustand brachte, kann der Programmierer wieder den $\overline{RES}$-Eingang aktivieren und damit die CPU in den Monitor zurückbringen (Restartoperation).

Die Behandlung von Unterbrechungen (interrupts)

Um die Rechenzeit (Ausführungszeit von Programmen) abzukürzen, besitzen viele Microprozessoren die Möglichkeit, von außen in den Programmablauf einzugreifen, diesen zu unterbrechen.

Diese Unterbrechungen sollen aber kontrollierbar sein. Die CPU soll nach einer bestimmten Unterbrechungsroutine wieder in das ursprüngliche Programm zurückfinden und an diesem weiterarbeiten.

Die Unterbrechungstechnik wird meist bei Reglungs- und Überwachungsprozessen angewandt. Auch die Bedienung langsamer Peripherie läßt sich dadurch optimieren.

Mit relativ geringem Hardwareaufwand kann die Unterbrechungstechnik zur Fehlersuche (debugging) von Programmen mit Vorteil eingesetzt werden (Einzelschrittbetrieb). Der 6502 besitzt zwei Interrupteingänge.

Die NMI-Unterbrechung

Wird der $\overline{\text{NMI}}$-Eingang (non maskable interrupt) aktiviert, findet eine Unterbrechung des aktuellen Programmablaufs statt, ob nun das I-Flag gesetzt ist oder nicht.

Die CPU arbeitet den Befehl ab, in dem sie sich gerade befindet. Dann legt sie den Programmzählerinhalt (Adresse des nächsten Befehls) und den Inhalt des Statusregisters in den Stack ab. Nun holt sie die neue Adresse aus dem Adressvektor NMI (FF FA und FF FB) in den Programmzähler. Das Interruptflag (I) wird gesetzt und sie beginnt mit der Bedienung der neuen Interruptroutine.

Die IRQ-Unterbrechung

Wird der $\overline{\text{IRQ}}$-Eingang (interrupt request) aktiviert, findet nur dann eine Programmunterbrechung statt, wenn das I-Flag gelöscht ist. Durch Setzen des I-Flags kann eine Unterbrechung unterbunden werden. Wurde eine Unterbrechung akezeptiert, so geschieht das eben Beschriebene. Das I-Flag wird natürlich auch gesetzt.

Neue Interruptanforderungen werden nicht mehr bedient (außer einem NMI-Impuls).

Soll die CPU aber doch noch weitere Interruptaufforderungen akzeptieren, muß der Programmierer durch Einfügen eines CLI-Befehls in die Unterbrechungsroutine dies ermöglichen.

Die Datensicherung

Da die CPU nach der Ausführung der Unterbrechung in das unterbrochene Programm zurückfinden und genau dort weiterarbeiten soll, muß für eine Datensicherung aller CPU-Register gesorgt werden, die von der Interruptroutine beeinflußt werden.

Am Anfang einer Interruptroutine muß also ein kleines „Sicherungsprogramm" stehen. Meist wird man die Registerinhalte in den Stack ablegen.

```
PHA     IMP   A nach Stack
TXA     IMP
PHA     IMP   X nach Stack
TYA     IMP
PHA     IMP   Y nach Stack
```

Die Rückkehr aus einer Interruptroutine

Am Ende eines Unterbrechungsprogramms muß für eine entsprechende Rückkehr in das unterbrochene Programm gesorgt werden.

Zuerst muß der Programmierer für eine Wiederherstellung der CPU-Registerinhalte sorgen.

```
PLA     IMP   Stack nach Y
TAY     IMP
PLA     IMP   Stack nach X
TAX     IMP
PLA     IMP   Stack nach A
```

Anschließend kommt der Befehl RTI.

RTI: Kehre aus einem Interruptprogramm zurück

```
RTI     IMP   40
```

Was bewirkt nun der Befehl RTI?
Das Prozessorstatusregister wird aus dem Stack geholt. Dabei wird natürlich das I-Flag gelöscht. Anschließend wird der Programmzähler mit der Rücksprungadresse aus dem Stack geladen. Nun befindet sich die CPU wieder im unterbrochenen Programm und die Register haben ihre ursprünglichen Inhalte.

Der Befehl BRK

Der 6502 besitzt einen Befehl, der eine IRQ-Unterbrechung erzeugt. Lädt die CPU den Operationscode des Befehls BRK, führt sie eine Unterbrechungsroutine aus und springt an die durch den IRQ-Vektor gekennzeichnete Speicherstelle. Der BRK-Befehl reagiert nicht auf das I-Flag.

BRK IMP 00

Der BRK-Befehl erlaubt dem Programmierer, in seinem Anwenderprogramm Unterbrechungspunkte anzulegen. Dadurch kann er den Ablauf seines Programms kontrollieren.

R6500 MICROPROCESSOR INSTRUCTION SET – ALPHABETIC SEQUENCE

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-Or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift Right One Bit (Memory or Accumulator)
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

INSTRUCTION ADDRESSING MODES AND RELATED EXECUTION TIMES (in clock cycles)

	Accumulator	Immediate	Zero Page	Zero Page, X	Zero Page, Y	Absolute	Absolute, X	Absolute, Y	Implied	Relative	(Indirect, X)	(Indirect), Y	Absolute Indirect
ADC		2	3	4		4	4*	4*			6	5*	
AND		2	3	4		4	4*	4*			6	5*	
ASL	2		5	6		6	7						
BCC										2**			
BCS										2**			
BEQ										2**			
BIT			3			4							
BMI										2**			
BNE										2**			
BPL										2**			
BRK													
BVC										2**			
BVS										2**			
CLC									2				
CLD									2				
CLI									2				
CLV									2				
CMP		2	3	4		4	4*	4*			6	5*	
CPX		2	3			4							
CPY		2	3			4							
DEC			5	6		6	7						
DEX									2				
DEY									2				
EOR		2	3	4		4	4*	4*			6	5*	
INC			5	6		6	7						
INX									2				
INY									2				
JMP						3							5
JSR						6							
LDA		2	3	4		4	4*	4*			6	5*	
LDX		2	3		4	4		4*					
LDY		2	3	4		4	4*						
LSR	2		5	6		6	7						
NOP									2				
ORA		2	3	4		4	4*	4*			6	5*	
PHA									3				
PHP									3				
PLA									4				
PLP									4				
ROL	2		5	6		6	7						
ROR	2		5	6		6	7						
RTI									6				
RTS									6				
SBC		2	3	4		4	4*	4*			6	5*	
SEC									2				
SED									2				
SEI									2				
STA			3	4		4	5	5			6	6	
STX			3		4	4							
STY			3	4		4							
TAX									2				
TAY									2				
TSX									2				
TXA									2				
TXS									2				
TYA									2				

* Add one cycle if indexing across page boundary

** Add one cycle if branch is taken, Add one additional if branching operation crosses page boundary

ØØ - BRK

Ø1 - ORA - (Indirect,X)

Ø2 - Future Expansion

Ø3 - Future Expansion

Ø4 - Future Expansion

Ø5 - ORA - Zero Page

Ø6 - ASL - Zero Page

Ø7 - Future Expansion

Ø8 - PHP

Ø9 - ORA - Immediate

ØA - ASL - Accumulator

ØB - Future Expansion

ØC - Future Expansion

ØD - ORA - Absolute

ØE - ASL - Absolute

ØF - Future Expansion

1Ø - BPL

11 - ORA - (Indirect),Y

12 - Future Expansion

13 - Future Expansion

14 - Future Expansion

15 - ORA - Zero Page,X

16 - ASL - Zero Page,X

17 - Future Expansion

18 - CLC

19 - ORA - Absolute,Y

1A - Future Expansion

1B - Future Expansion

1C - Future Expansion

1D - ORA - Absolute,X

1E - ASL - Absolute,X

1F - Future Expansion

2Ø - JSR

21 - AND - (Indirect,X)

22 - Future Expansion

23 - Future Expansion

24 - BIT - Zero Page

25 - AND - Zero Page

26 - ROL - Zero Page

27 - Future Expansion

28 - PLP

29 - AND - Immediate

2A - ROL - Accumulator

2B - Future Expansion

2C - BIT - Absolute

2D - AND - Absolute

2E - ROL - Absolute

2F - Future Expansion

3Ø - BMI

31 - AND - (Indirect),Y

32 - Future Expansion

33 - Future Expansion

34 - Future Expansion

35 - AND - Zero Page,X

36 - ROL - Zero Page,X

37 - Future Expansion

38 - SEC

39 - AND - Absolute,Y

3A - Future Expansion

3B - Future Expansion

3C - Future Expansion

3D - AND - Absolute,X

3E - ROL - Absolute,X

3F - Future Expansion

40 - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
50 - BVC
51 - EOR - (Indirect),Y
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion

60 - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
70 - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion

8Ø - Future Expansion

81 - STA - (Indirect,X)

82 - Future Expansion

83 - Future Expansion

84 - STY - Zero Page

85 - STA - Zero Page

86 - STX - Zero Page

87 - Future Expansion

88 - DEY

89 - Future Expansion

8A - TXA

8B - Future Expansion

8C - STY - Absolute

8D - STA - Absolute

8E - STX - Absolute

8F - Future Expansion

9Ø - BCC

91 - STA - (Indirect),Y

92 - Future Expansion

93 - Future Expansion

94 - STY - Zero Page,X

95 - STA - Zero Page,X

96 - STX - Zero Page,Y

97 - Future Expansion

98 - TYA

99 - STA - Absolute,Y

9A - TXS

9B - Future Expansion

9C - Future Expansion

9D - STA - Absolute,X

9E - Future Expansion

9F - Future Expansion

AØ - LDY - Immediate

A1 - LDA - (Indirect,X)

A2 - LDX - Immediate

A3 - Future Expansion

A4 - LDY - Zero Page

A5 - LDA - Zero Page

A6 - LDX - Zero Page

A7 - Future Expansion

A8 - TAY

A9 - LDA - Immediate

AA - TAX

AB - Future Expansion

AC - LDY - Absolute

AD - LDA - Absolute

AE - LDX - Absolute

AF - Future Expansion

BØ - BCS

B1 - LDA - (Indirect),Y

B2 - Future Expansion

B3 - Future Expansion

B4 - LDY - Zero Page,X

B5 - LDA - Zero Page,X

B6 - LDX - Zero Page,Y

B7 - Future Expansion

B8 - CLV

B9 - LDA - Absolute,Y

BA - TSX

BB - Future Expansion

BC - LDY - Absolute,X

BD - LDA - Absolute,X

BE - LDX - Absolute,Y

BF - Future Expansion

CØ - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
DØ - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP - Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion

EØ - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
FØ - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,X
FF - Future Expansion

4. Die Programmsammlung

Will man die Programmierung in einem Assemblercode (zum Beispiel dem des 6502) erlernen, so reicht nicht das Durchlesen eines entsprechenden, oft umfangreichen, Programmierhandbuches aus. Man muß unbedingt an vielen Beispielen die Methoden und Feinheiten der Programmiertechniken erlernen.

Dieses Kapitel will nun entsprechende Beispiele anbieten. Manche Beispiele haben nicht nur Übungscharakter, dazu sind sie zu umfangreich, sie sollen schon den Charakter eines größeren Anwenderprogrammes annehmen. Dieses Buch will kein systematisch abgeschlossenes Programmierlehrbuch sein, wie schon erwähnt wurde, sondern es will beim Erlernen des Programmierens im Maschinencode des 6502 helfen.

Der eine oder andere Leser wird sicherlich das Programmierlehrbuch (programming manual) des 6502 parallel lesen müssen, um Antworten auf eventuell auftretende Fragen zu finden.

Die Beispiele sind nach zwei verschiedenen Kriterien ausgesucht und angeordnet worden. Das erste Kriterium ist der sachliche Zusammenhang (ähnliche Problemstellungen). Das zweite Kriterium ist der Schwierigkeitsgrad. Es wurde versucht, die Beispiele innerhalb eines Problemkreises nach ansteigendem Schwierigkeitsgrad anzuordnen.

Derjenige Leser, der keine Programmiererfahrungen hat, sollte unbedingt die ausgesuchte Reihenfolge einhalten, da die Programmiertechniken immer bei den Beispielen ausführlich erläutert werden, bei denen sie das erste Mal Verwendung finden. Es ist nicht möglich und auch nicht sinnvoll, das erste und das letzte Beispiel in der gleichen Breite zu beschreiben.

Derjenige Leser, der Programmiererfahrungen hat, muß sich nicht unbedingt an die Reihenfolge halten, denn auf die wichtigsten Querzüge wird in jedem Beispiel, wenn nötig, hingewiesen. Die Erläuterungen zu jedem Beispiel wurden bewußt breit gehalten, um einen Nachteil vieler Programmsammlungen und Lehrbücher zu vermeiden.

Ein jedes Beispiel beginnt mit der Problemstellung und dessen Analyse. Es folgt die Entwicklung des Lösungsweges und die Abbildung des Ablaufplanes (Flußdiagramms). Das Programm wird dann im Assemblercode formuliert, wenn dies notwendig erscheint, ansonsten wird es gleich im Maschinencode angegeben.

Die Syntax (formaler Aufbau) des Assemblers sei hier kurz beschrieben.
Ein Programm besteht aus:

1. Sprungmarke (wenn erforderlich),
2. Operationscode + Adresscode,
3. Operand

Beispiel:

Marke	Op.Code	Adr.Code	Operand
M1	LDA	IM	F0
	STA	ZP	05 (Adresse des Operanden)
	JMP	AB	M1 (Sprungmarke)

Die Marken können beliebige Zeichenfolgen sein. Der Operations- und der Adresscode entsprechen der Mnemonic aus Kapitel 3.3.

Das Listing des Programmes im Hexcode hat ein etwas ungewöhnliches Aussehen, da hinter dem Operationsbyte der Assemblercode des Befehls angegeben wird. So ist das Listing ähnlich leicht lesbar wie die Formulierung des Programmes im Assemblercode.
Ein Beispiel einer Programmzeile:

Adresse	Befehlsbyte	Mnemonic	1. Adressb.	2. Adressbyte
0200	20	JSR AB	00	10

Abschließend wird ein Speicherauszug des Programms im Hexcode angegeben, allerdings nicht bei ganz einfachen und kurzen Programmen.

4.1. Grundprogramme

Unter Grundprogrammen verstehen wir Programme, die ein relativ leicht überschaubares Problem lösen und die häufig Grundeinheiten komplexerer Programme sind.

Wir werden hier einfache mathematische Grundprogramme, Grundprogramme einer Steuerungsanlage und Grundprogramme zur Bedienung peripherer Ein- und Ausgabeeinheiten kennenlernen.

4.1.1 Mathematische Grundprogramme

Die ersten beiden Beispiele sind reine lineare Programme. Schon im vierten Beispiel stelle ich die Programmierung von Schleifen vor. Dies ist eine sehr wichtige Programmiertechnik, ohne die die Programme sehr unübersichtlich groß würden.

1. Addition (dual)
Die Problemstellung und Analyse:

Es sollen zwei Zahlen dual addiert werden.
Wie läßt sich das verwirklichen? Die CPU besitzt einen Additionsbefehl (ADC), der folgende Funktion hat: A + M + C → A.
Man könnte die Aufgabenstellung so formulieren:
Schreibe in eine bestimmte Speicherzelle Z1 eine Zahl (dual) und in eine andere Zelle Z2 eine weitere Zahl. Die Summe dieser Zahlen soll in die Zelle Z3 gespeichert werden. Diese Zelle kann man nach Programmablauf lesen.
Die Aufgabenstellung ist so ausführlich formuliert, daß der Lösungsweg leicht formulierbar ist.

Der Lösungsweg:

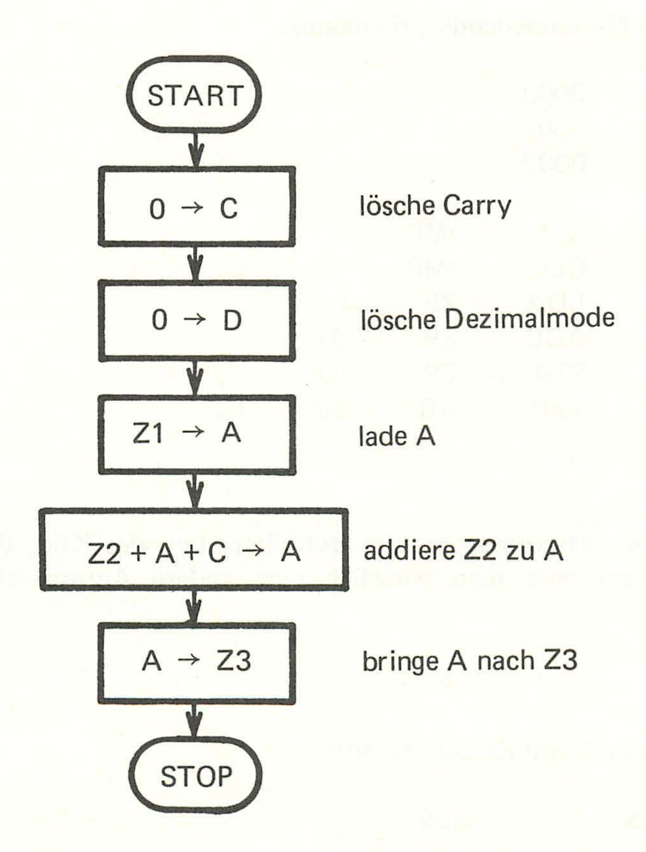
Beim ADC-Befehl wird der Inhalt einer Speicherzelle mit einem evtl. vorhandenen Übertrag (C = 1) zum Akkumulatorinhalt addiert. Wir müssen also zuerst das C-Flag löschen (C = 0), es könnte ja gesetzt sein.

Anschließend müssen wir beachten, daß die CPU zwei Additionsweisen kennt (die duale oder die dezimale). Da wir uns für die duale Addition

entschieden, muß der Dualmode eingeschaltet werden (D = 0). Diese beiden Operationen könnte man als Vorbereitung bezeichnen.

Nun beginnt der eigentliche Additionsvorgang. Der erste Summand muß in A geladen werden, dann kann der zweite Summand hinzuaddiert werden. Die Summe steht nun in A. Ein Transportbefehl bringt die Summe nach Z3.
Der Ablaufplan zeigt alles noch einmal in einer geeigneten Kurzschreibweise.

Der Ablaufplan:



Die Übersetzung des Ablaufplanes in ein Programm im Assemblercode wird uns nicht schwerfallen.

Über die Lage des Programms im Speicher sollte man sich auch schon Gedanken machen. Man könnte es in beliebige Speicherbereiche ablegen. Üblich ist es, die Daten in die 0. Seite zu legen und das Programm ab Adresse 0200 abzuspeichern. Die Seite 0000 bis 00FF ist ja für den Stackbereich reserviert. Daher kann man hier nur sehr kleine Programme unterbringen, wenn man diesen Speicherbereich unbedingt benötigt.

Da das Programm so kurz ist, formulieren wir es gleich im Maschinencode.

Das Programm im Maschinencode (Hexcode):

Adressen:	Z1	0000			
	Z2	0001			
	Z3	0002			
02 00	18	CLC	IMP		
01	D8	CLD	IMP		
02	A5	LDA	ZP	00	
04	65	ADC	ZP	01	
06	85	STA	ZP	02	
08	4C	JMP	AB	00	1C

1C 00 ist die Eintrittsadresse für das Betriebssystem des KIM. Bei anderen Computern muß man natürlich eine andere Adresse einsetzen.

Probeläufe:

1. Schreiben wir in Z1 und Z2 zum Beispiel:

		hex		dual		
	Z1	15	=	0001 0101	21	(dezimal)
+	Z2	4E	=	0100 1110	+ 78	
=	Z3	63	=	0110 0011	99	

2. Schreiben wir in Z1 und Z2 zum Beispiel:

	Z1	E5	=	1110 0101	229	
+	Z2	37	=	0011 0111	+ 55	
=	Z3	1C	=	0001 1100	248	(größer als 256)

Es entsteht ein Überlauf (Übertrag, C = 1)

2. Subtraktion (dual)

Haben wir das Additionsprogramm verstanden und genügend analysiert, so ist die Verwirklichung eines Subtraktionsprogrammes sehr einfach. Wir können das gleiche Schema des Ablaufplanes verwenden und müssen nur zwei Befehle abändern.

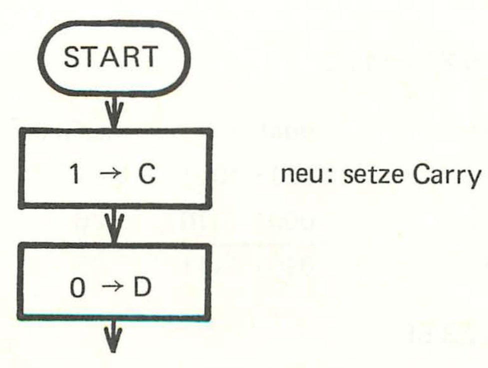
Die Problemstellung:

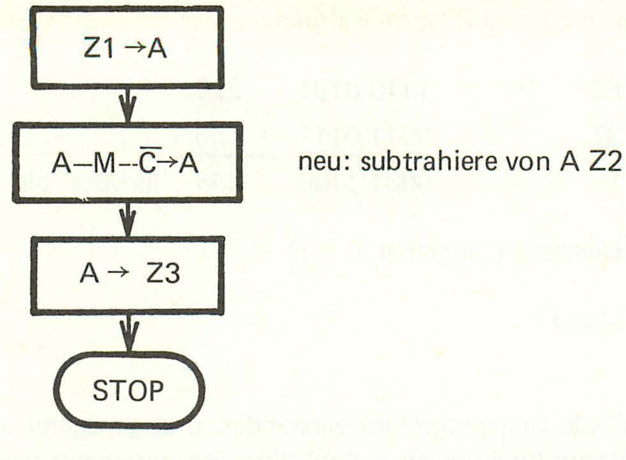
Die Zahl in Z2 soll von der Zahl in Z1 subtrahiert und die Differenz in Z3 gespeichert werden.

Der Lösungsweg:

Die CPU besitzt den Subtraktionsbefehl SBC (A–M–$\overline{C}$ → A). Es muß jetzt auch auf die Stellung des C-Flags geachtet werden. Das C-Flag muß gesetzt (C = 1) sein. Ebenfalls muß die duale Betriebsart eingestellt werden.

Der Ablaufplan:





Wir formulieren das Programm nun auch gleich im Maschinencode.

Programm im Maschinencode (Hexcode):

Adressen:	Z1	00	00
	Z2	00	01
	Z3	00	02

02 00	38	SEC	IMP		
01	D8	CLD	IMP		
02	A5	LDA	ZP	00	
04	E5	SBC	ZP	01	
06	85	STA	ZP	02	
08	4C	JMP	AB	00	1C

Probeläufe:

1. Schreiben wir in Z1 und Z2

		hex		dual	dezimal
	Z1	79	=	0111 1001	121
–	Z2	1A	=	0001 1010	– 26
=	Z3	5F	=	0101 1111	95

so erhalten wir in Z3 5F.

2. Schreiben wir in Z1 und Z2

		hex		dual	dezimal
	Z1	1A	=	0001 1010	26
–	Z2	79	=	0111 1001	– 121
=	Z3	C1	=	1010 0001	– 95

C = 0

Die Differenz ist nun eine negative Zahl, was man am gelöschten C-Flag erkennen kann. Die negative Zahl –95 wird im 2er Komplement dargestellt.

An dieser Stelle kann nicht näher auf die Arithmetik mit negativen Zahlen eingegangen werden. Hierzu muß auf die entsprechende Spezialliteratur verwiesen werden.

3. Berechnung relativer Adressen

Unser Computer soll uns die etwas mühselige Berechnung der relativen Adressen bei bedingten Verzweigungen abnehmen. Das hier vorgestellte Programm wird uns sehr nützlich sein und könnte ein Unterprogramm eines Assemberprogramms werden.

Eigentlich ist auch dieses Programm linear aufgebaut. Zum Schluß, bei der Fehleranalyse müssen allerdings einige bedingte Verzweigungen eingeführt werden. Ebenfalls neu ist hier die Verwendung einer doppelt genauen Subtraktion.

Die Problemstellung:

Es wird die Startadresse, die Adresse des dem Sprungbefehl folgenden Befehlsbytes, und die Sprungzieladresse in den Datenspeicher eingelesen. Der Computer soll nun das Adressbyte (relative Adresse– des Sprungbefehls berechnen. Da nur maximal 127 Bytes vorwärts und 128 Bytes zurück übersprungen werden können, soll der Computer auch eine Fehlermeldung abgeben, wenn die Start- und Zieladresse zu weit entfernt sind.

Die Problemanalyse:

Man kann das 2er-Komplement einer Dualzahl als negative Zahl verstehen, wie in Kapitel 2.1.1 vorgestellt wurde. Den Rückwärtssprung um 4 Bytes könnte man durch die Zahl –4 ausdrücken, was ja auch wirklich geschieht. Wir müssen bei Rückwärtssprüngen als Adressbyte das 2er-Komplement der Bytezahl (Sprungweite) einsetzen.

Die Berechnung der Sprungweite (positiv oder negativ) ist sehr einfach. Wir subtrahieren die Startadresse von der Zieladresse. Dabei muß eine 16 bit Subtraktion durchgeführt werden. Das sei an zwei Beispielen veranschaulicht.

Vorwärtssprung:

	Hex	dual	dezimal
Zieladresse:	03 51	0000 0011 0101 0001	
Startadresse:	02 F7	0000 0010 1111 0111	
Differenz:	00 5A	0000 0000 0101 1010	+90
	C = 1		

Die relative Adresse ist 5A.

Rücksprung:

	hex	dual	dezimal
Zieladresse:	02 F7	0000 0010 1111 0111	
Startadresse:	03 51	0000 0011 0101 0001	
Differenz:	FF A6	1111 1111 1010 0110	--90
		C = 0	

Das 2. Byte der Differenz beinhaltet nun das 2er-Komplement der Sprungweite. In beiden Fällen ist es die gesuchte relative Adresse.
Was geschieht nun, wenn die Sprungweite zu groß wird? Betrachten wir zwei Beispiele zum Vorwärtssprung.

1. Beispiel:

	hex	dual	dezimal
Zieladresse:	03 51	0000 0011 0101 0001	
Startadresse:	02 41	0000 0010 0100 0001	
Differenz:	01 10	0000 0001 0001 0000	+ 272

C = 1

2. Beispiel:

	hex	dual	dezimal
Zieladresse:	03 51	0000 0011 0101 0001	
Startadresse:	02 61	0000 0010 0110 0001	
Differenz:	00 B0	0000 0000 1011 0000	+ 176

C = 1

Wird die Sprungweite 127 überschritten, so wird das 1. Byte der Differenz ungleich 00 oder das 8. Bit des 2. Bytes ist 1.

Nun zwei Beispiele zum Rückwärtssprung.

1. Beispiel:

	hex	dual	dezimal
Zieladresse:	02 41	0000 0010 0100 0001	
Startadresse:	03 51	0000 0011 0101 0001	
Differenz:	FE F0	1111 1110 1111 0000	– 272

C = 0

2. Beispiel:

	hex	dual	dezimal
Zieladresse:	02 61	0000 0010 0110 0001	
Startadresse:	03 51	0000 0011 0101 0001	
Differenz:	FF 10	1111 1111 0001 0000	−176

C = 0

Wir fassen zusammen. Wird bei einem Rückwärtssprung die Sprungweite 128 überschritten, so wird das 1. Byte der Differenz ungleich FF oder das 8. Bit des 2. Bytes ist 0.

Die Entscheidung, ob ein Vorwärts- oder Rücksprung vorliegt, kann durch Abfragen des Inhaltes von C geschehen.

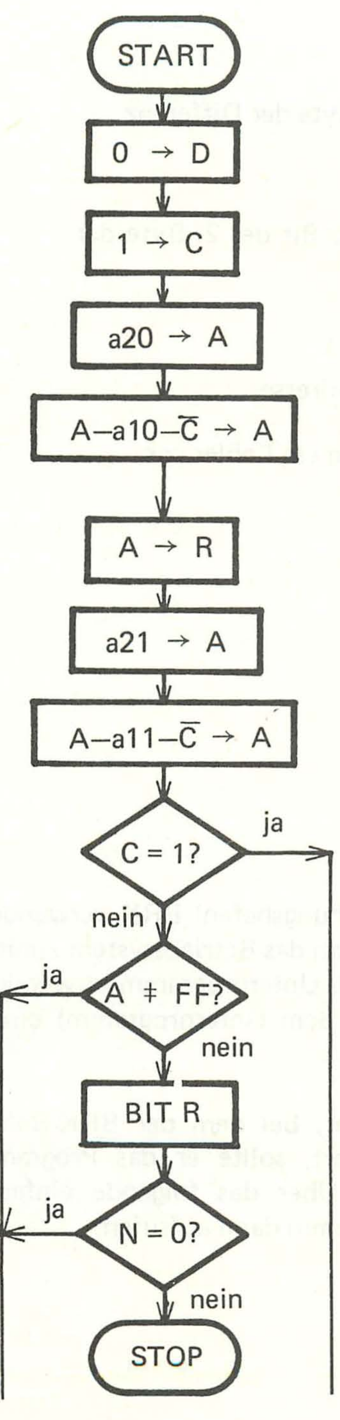
Der Lösungsweg:

Nach dieser eingehenden Diskussion liegt der Lösungsweg auf der Hand. Da eine Fehleranalyse durchgeführt werden soll, muß eine Subtraktion mit 2byte-Zahlen ausgeführt werden. Das 2. Byte der Differenz beinhaltet die aktuelle relative Sprungadresse und das 1. Byte ist für die Fehleranalyse notwendig. Wenn ein Fehler vorliegt, soll die relative Adresse 00 gesetzt werden, was sonst nicht vorkommen kann.

Eine 2byte-Subtraktion ist mit Hilfe des C-Flags leicht programmiert. Das C-Flag wird gesetzt und dann die niederwertigsten 8 Bits subtrahiert. Ein eventueller Übertrag wird in C gespeichert (C = 0). Bei der folgenden Subtraktion der höherwertigen 8 Bits wird dieser Überlauf dann automatisch beachtet.

Der Ablaufplan:

Adressenbelegung:	Startadresse	2. Byte:	00 01	a10
		1. Byte:	00 02	a11
	Zieladresse	2. Byte:	00 03	a20
		1. Byte:	00 04	a21
	rel. Adresse:		00 00	R



duale Betriebsart

Vor einer Subtraktion muß das C-Flag gesetzt werden.

Das niederwertige Byte des Minuenden wird in A geladen.

Das niederwertige Byte des Subtrahenden wird subtrahiert und die Differenz steht in A. Wenn ein Überlauf auftritt wird C = 0.

Die Differenz ist die relative Adresse.

Das höherwertige Byte des Minuenden wird in A geladen.

Die Differenz der höheren Byte steht in A, der Überlauf wurde beachtet.

Es folgt nun die Fehleranalyse.

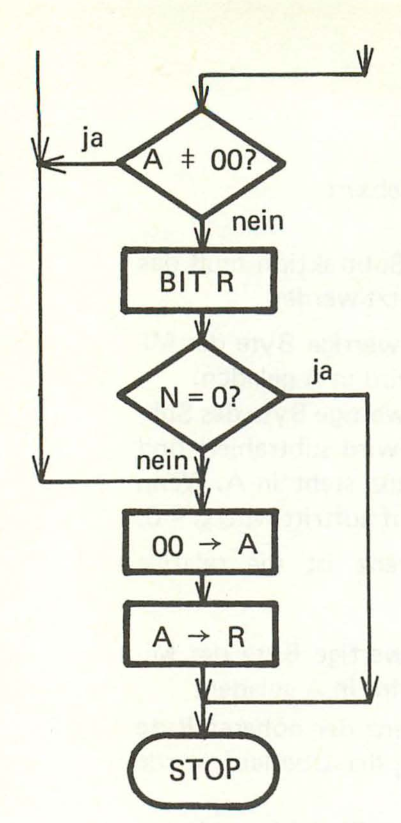
Ist die Differenz positiv?

Ist das erste Byte nicht FF?

Wir verwenden den BIT-Befehl, um das 8. Bit des 2. Byte der Differenz zu testen.

Ist das 8. Bit gleich 0?

Es liegt eine zulässige Adresse vor.



Ist das 1. Byte der Differenz nicht 00?

Test des 8. Bit des 2. Byte der Differenz

zulässige Adresse

Es liegt nun ein Fehler vor.

Als „STOP"-Befehl wird der Unterbrechungsbefehl BRK verwendet, damit die Kontrolle des Microprozessors an das Betriebssystem zurückgegeben wird. Will man das Programm als Unterprogramm verwenden, muß der RTS-Befehl (Rücksprung aus dem Unterprogramm) eingesetzt werden.

Besitzt der Leser einen Microcomputer, bei dem der BRK-Befehl nicht in das Betriebssystem zurückführt, sollte er das Programm gleich als Unterprogramm aufbauen. Über das folgende einfache Hauptprogramm kann er das Unterprogramm dann aufrufen.

```
     JSR  AB  Unterprogramm
M1   JMP  AB  M1          (Stopschleife)
```

Das Programm im Maschinencode (Hexcode):

```
0200  d8  cld imp
0201  38  sec imp
0202  a5  lda zp  03
0204  e5  sbc zp  01
0206  85  sta zp  00
0208  a5  lda zp  04
020a  e5  sbc zp  02
020c  b0  bcs r   0a
020e  c9  cmp im  ff
0210  d0  bne r   0c
0212  24  bit zp  00
0214  10  bpl r   08
0216  30  bmi r   0a
0218  d0  bne r   04
021a  24  bit zp  00
021c  10  bpl r   04
021e  a9  lda im  00
0220  85  sta zp  00
0222  00  brk imp
```

Der Speicherauszug:

```
0200 d8 38 a5 03 e5 01 85 00 a5 04 e5 02 b0 0a c9 ff
0210 d0 0c 24 00 10 08 30 0a d0 04 24 00 10 04 a9 00
0220 85 00 00
```

4. 12stellige BCD Addition

Eine Rechengenauigkeit von 8 bit ist recht gering. Wir können so nur Zahlen zwischen 0 und 255 darstellen. Wenn wir die Rechengenauigkeit erweitern, was mit Hilfe geeigneter Programme durchführbar ist, bleibt uns das recht aufwendige Geschäft der Zahlenumwandlung vom Dualcode in den Dezimalcode.

Das ist ein Grund für die Verwendung dual codierter Dezimalzahlen. Da der 6502 sogar einen entsprechenden Betriebszustand für die Rechnung im BCD-Code besitzt, liegt es nahe, bei Rechnungen mit höherer Genauigkeit im BCD-Code zu arbeiten.

Als Beispiele sollen hier und im folgenden Abschnitt die 12stellige Addition und Subtraktion vorgestellt werden. Diese Programme lassen sich einfach auf eine beliebig hohe Stellenzahl erweitern.

Als neue Programmiermethode wollen wir die Programmierung von Schleifen einsetzen. Dabei werden wir auch die indizierte Adressierung kennenlernen.

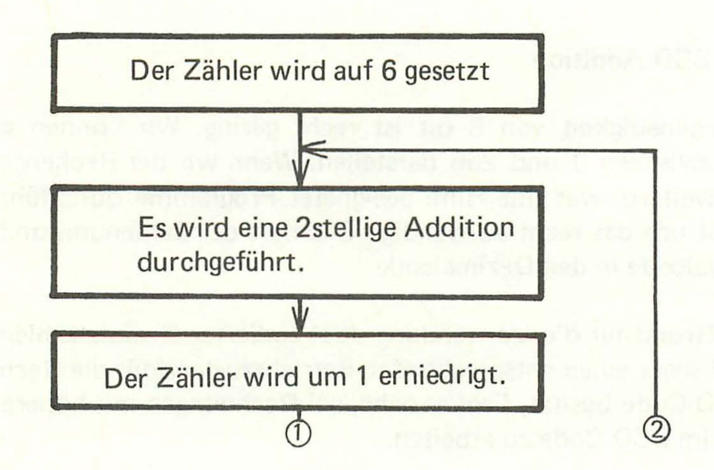
Die Problemstellung ist klar. Wir wollen uns gleich der Problemlösung zuwenden.

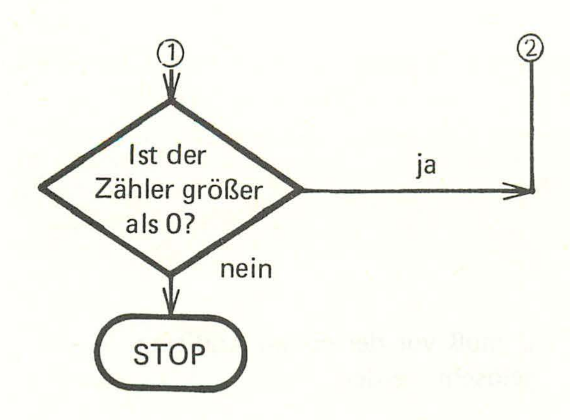
Die Problemlösung:

Für eine 12stellige BCD-Zahl benötigen wir 6 Speicherzellen, da ein Byte zwei Ziffern darstellen kann. Es müssen also sechs Additionen ausgeführt werden. Man könnte dies natürlich mit einem linearen Programm durchführen. Dann ist aber eine Erhöhung der Stellenzahl nicht einfach und das Programm würde einige Speicherzellen mehr beanspruchen.

Da sechs gleichartige Operationen durchgeführt werden, liegt es nahe, einen Zähler einzuführen, der die einzelnen Operationen mitzählt und nach sechs Operationsdurchführungen das Programm beendet.

Dies geschieht nach folgendem Schema:





Bei der immer wiederkehrenden Addition muß darauf geachtet werden, daß jedesmal zwei andere Stellenpaare verknüpft werden. Diese Stellenpaare müssen also auf geeignete Weise durch den Zähler adressiert werden.

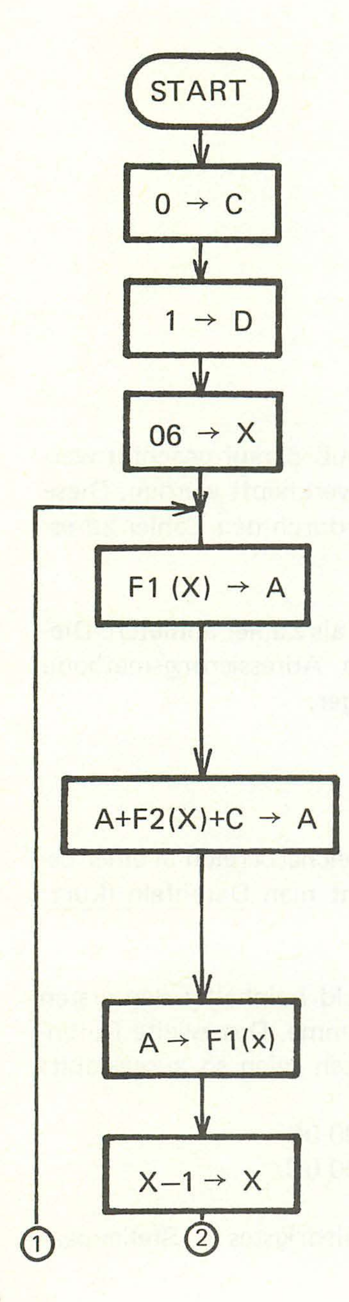
Der 6502 besitzt zwei Indexregister, die sich als Zähler anbieten. Diese Indexregister dienen bei der indirekten Adressierungsmethode (Kap. 3.2.3) auch als Indizes oder Speicherzeiger.

Daten, die in einem zusammenhängenden Speicherbereich in einer bestimmten Reihenfolge angeordnet sind, nennt man <u>Datenfeld (kurz: Feld)</u>.

Wir legen zwei Datenfelder an. Das erste Feld beinhaltet den ersten Summanden und nach der Rechnung die Summe. Das zweite Datenfeld enthält den 2. Summanden. Die Adressen seien so ausgewählt:

1. Summand u. Summe:	F1	00 01	00 06
2. Summand:	F2	00 07	00 0C
		höchstes	niedrigstes Stellenpaar

Der Ablaufplan



C muß vor der ersten Addition gelöscht werden.

Wir verwenden die dezimale Betriebsart.

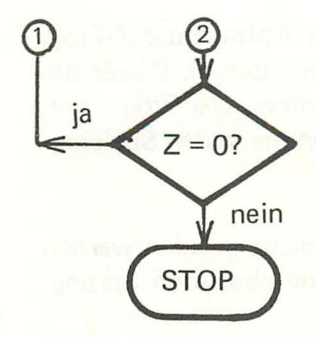
Das X-Register wird als Zähler verwendet und mit 6 geladen.

Das X-te Stellenpaar des 1. Feldes wird in den Akkumulator geholt. Wenn X = 6 ist, wird also das Byte 06 der 0. Seite geladen. Das sind die ersten zwei Stellen des ersten Summanden.

Das X-te Stellenpaar des zweiten Feldes wird hinzuaddiert. Das C-Flag beinhaltet den Überlauf, falls vorhanden. Wenn X=6 ist, wird das Byte 0C der 0. Seite addiert.

Die Summe wird in die entsprechende Stelle des 1. Feldes abgespeichert.

Der Zähler wird decrementiert.



Ist das Z-Flag nicht gesetzt?

Wenn X = 0, ist Z = 1. Dann soll das Programm beendet werden.

Die Übersetzung dieses Ablaufplanes in ein Programm wird uns nicht schwerfallen. Wir verwenden bei den Speicherlese- und -schreibbefehlen und dem Additionsbefehl die ZPX-indizierte Adressierung, da die Felder in der 0. Seite liegen.

In diesem Programm tritt ein Rücksprungbefehl auf, dessen Adressrechnung wir hier durchführen.

	hex	dual
Zieladresse:	02 04	0000 0010 0000 0100
Startadresse:	02 0D	0000 0010 0000 1101
Differenz:	FF F7	1111 1111 1111 0111

Die relative Adresse ist also F7.

Ein Probelauf

Wir laden in das Feld F1 (0. Seite 06 bis 01) einen beliebigen Summanden und in das Feld (0. Seite 0C bis 07) einen anderen. Das Programm starten wir ab Adresse 0200. Das C-Flag wird gelöscht, damit bei der ersten Addition nicht zufällig eine 1 hinzuaddiert wird. Da X = 6 beim ersten Schleifendurchgang, wird das Byte 0C zum Byte 06 addiert und die Summe in das Feld F1 an die Stelle 06 gebracht. X wird anschließend decrementiert. Da die Abfrage des Z-Flags negativ ausfällt, springt die CPU an die Programmstelle 0204. Jetzt wird das Byte 0B zum Byte 05 addiert und die Summe in 05 abgespeichert. Anschließend decrementiert X und die CPU springt wieder an die Stelle 0204. Beim 6. Durchgang der Schleife beinhaltet X die Zahl 1.

Wird jetzt decrementiert, fällt die anschließende Abfrage des Z-Flags positiv aus und damit gelangt die CPU zum BRK-Befehl. Dieser Befehl bewirkt einen Rücksprung in das Betriebssystem. Der Programmablauf ist damit beendet und wir können die Summe in den Speicherzellen 01 bis 06 finden.

Dieses Programm kann in beliebige Speicherbereiche geladen werden und ist ohne Veränderungen lauffähig, da keine absoluten Sprungziele vorkommen.

Das Programm im Maschinencode:

```
0200   18   clc imp
0201   f8   sed imp
0202   a2   ldx im  06
0204   b5   lda zpx 00
0206   75   adc zpx 06
0208   95   sta zpx 00
020a   ca   dex imp
020b   d0   bne r   f7
020d   00   brk imp
```

Der Speicherauszug:

```
0200   18 f8 a2 06 b5 00 75 06 95 00 ca d0 f7 00
```

5. 12stellige BCD Subtraktion

Da dieses Programmbeispiel sehr der Addition ähnelt, sind ausführliche Erläuterungen unnötig.

Wir fügen einen Subtraktionsbefehl ein und setzen vor der ersten Subtraktion das C-Flag.

Der Minuend und die Differenz befinden sich im 1. Feld (F1). Der Subtrahend soll in das 2. Feld (F2) geladen werden.

Es muß natürlich beachtet werden, daß der Minuend immer größer oder gleich dem Subtrahenden ist, was eventuell vorher in einem Unterprogramm überprüft werden kann.

Das Programm im Maschinencode:

```
0200  f8  sed imp
0201  38  sec imp
0202  a2  ldx im  06
0204  b5  lda zpx 00
0206  f5  sbc zpx 06
0208  95  sta zpx 00
020a  ca  dex imp
020b  d0  bne r   f7
020d  00  brk imp
```

Der Speicherauszug:

```
0200  f8 38 a2 06 b5 00 f5 06 95 00 ca d0 f7 00
```

6. Multiplikation (dual)

Der 6502 besitzt, wie die meisten anderen Microprozessoren, keinen Multiplikationsbefehl. Will man in bestimmten Anwendungen eine 8 bit Multiplikation durchführen, muß man auf ein geeignetes Unterprogramm zurückgreifen. Zwei solche Programme sollen hier vorgestellt werden.

Man könnte als Multiplikationsalgorithmus den sehr einfachen Additionsalgorithmus, der die Multiplikation auf eine mehrfache Addition zurückführt, verwenden. Dieser wird im ersten Beispiel vorgestellt. Er ist leicht programmierbar, besitzt aber den Nachteil, daß er recht langsam ist.

Der zweite Algorithmus verwendet das bekannte Verfahren der „schriftlichen Multiplikation". Er ist nicht so einfach programmierbar, aber bedeutend schneller. Will man mehrfache Multiplikationen durchführen oder ist die Rechenzeit entscheidend (z. B. in numerischen Steuerungsanlagen), so wird man sicherlich auf den zweiten Algorithmus zurückgreifen. Das zweite Programmbeispiel stellt ihn vor.

6.1 Der Additionsalgorithmus

6 x 3 läßt sich so berechnen: 6 + 6 + 6 = 18. Wir addieren schlicht die Zahl 6 3 mal.
6 x 128 = 6 + 6 + ... + 6 (128 mal)

oder

= 128 + 128 + ... + 128 (6 mal)

Die Anzahl der Additionen hängt also wesentlich von der Größe der Faktoren ab.

Man kann die Rechenzeit optimieren, wenn man vor Beginn der Rechnung die Faktoren nach ihrer Größe anordnet.

Die Problemstellung:

Es sollen zwei vorzeichenlose 8 bit Zahlen multipliziert werden.

Die Problemlösung:

Wir ordnen die Faktoren ihrer Größe nach an und verwenden das Prinzip der Mehrfachaddition.
Die folgenden Speicherzellen werden in der 0. Seite reserviert:

1. Faktor	F1	01
2. Faktor	F2	02
Hilfszelle	H	03
Produkt	P	00

Wie soll der Vergleich der Faktoren erfolgen?
Wir führen eine Subtraktion aus und entscheiden dann, dem Vorzeichen der Differenz entsprechend. Daher müßten wir eigentlich den Zahlenbereich auf −127 bis + 127 einschränken. Das ist eine unnötige Einschränkung, zumal wir vorzeichenlos rechnen wollen.

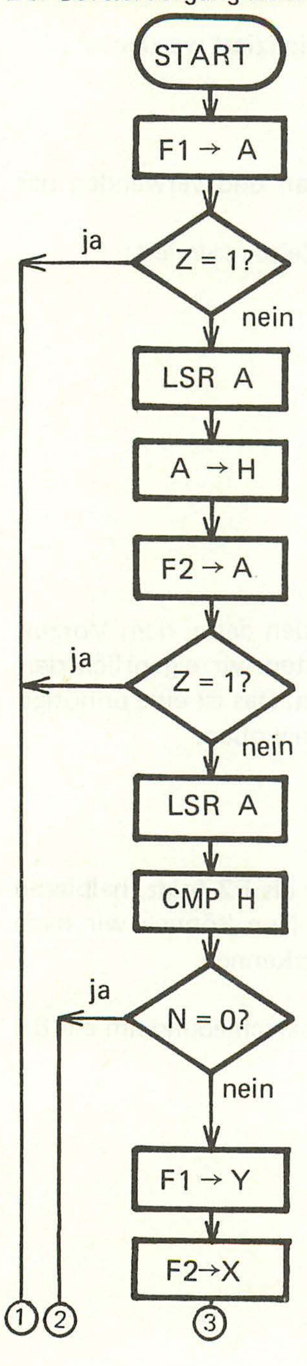
Da aus F1/2 größer als F2/2 auch F1 größer als F2 folgt, halbieren wir die Faktoren vor dem Vergleich einfach. Nun können wir nach der Subtraktion sicher eine negative Differenz erkennen.

Man halbiert eine Dualzahl durch eine Stellenverschiebung um ein Bit nach rechts. Ein Beispiel:

$$28_{10} = 11100_2$$
$$14_{10} = 01110_2$$

Gleichzeitig überprüfen wir, ob ein Faktor 0 ist.

Der Sortiervorgang könnte also so ablaufen:



Der 1. Faktor wird in den Akkumulator geladen.

Ist er Null?

Division durch 2.

Er wird im Hilfsspeicher zwischengespeichert.

Der 2. Faktor wird geladen.

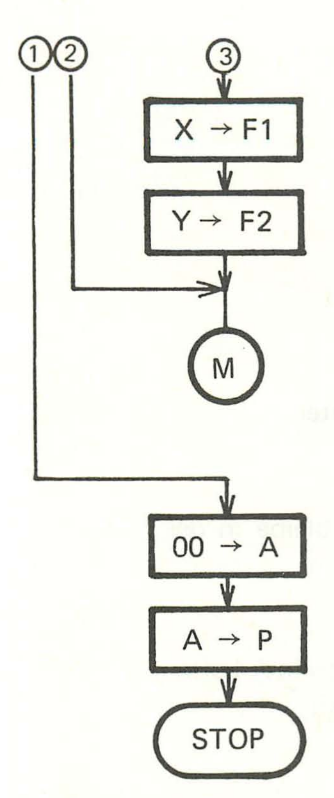
Ist er Null?

Division durch 2.

Der Inhalt von A wird mit dem von H verglichen. Es findet eine Subtraktion A–H statt. Ist das Ergebnis nicht negativ, so kann auf eine Vertauschung verzichtet werden.
Der Vertauschungsprozess:

Der erste Faktor wird in das Y-Register geladen.

Der zweite Faktor wird in das X-Register geladen.



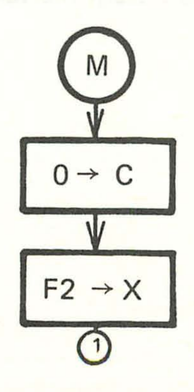
Nun wird umgeordnet.

Sprung in das eigentliche Multiplikationsprogramm.

Wenn ein Faktor Null ist, wird eine 0 in die Ergebnisspeicherzelle (P) geladen und das Programm ist schon abgeschlossen.

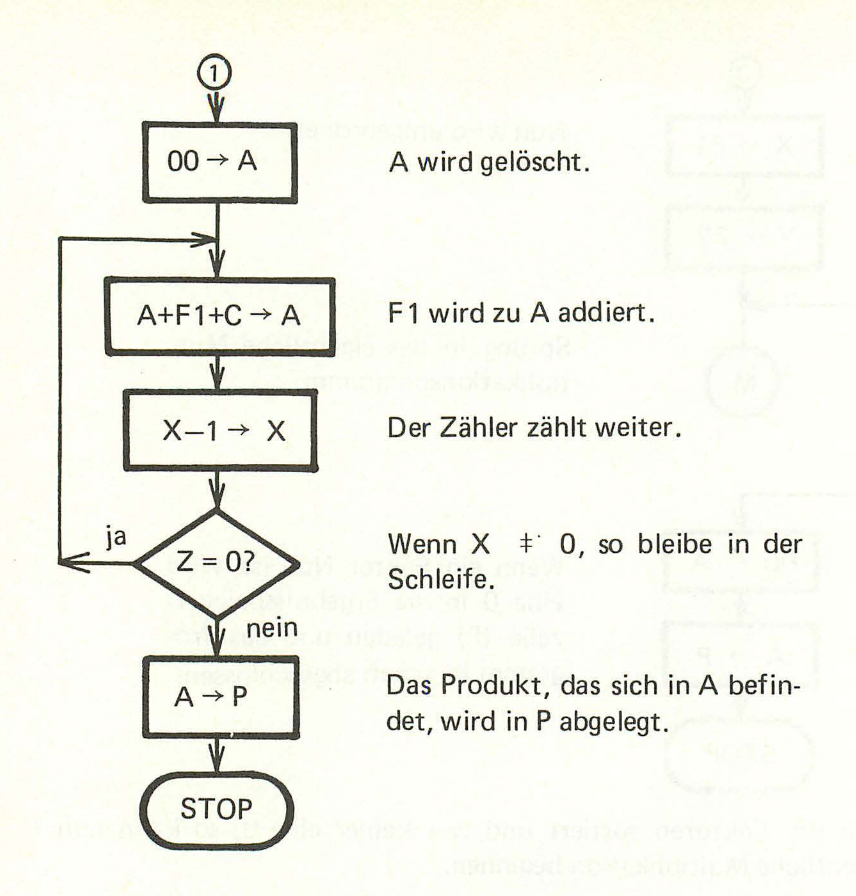
Wurden die Faktoren sortiert und war keiner eine 0, so kann nun die eigentliche Multiplikation beginnen.

Wir nehmen an, daß die CPU sich im dualen Rechenmode befindet. Es wird natürlich in Schleifentechnik programmiert, der kleinere Faktor gibt die Anzahl der Schleifendurchläufe an.



Löschen des C-Flags

Das Indexregister X wird mit dem kleineren Faktor geladen.



Nun wird der Ablaufplan in den Maschinencode übersetzt. Wir verwenden als „STOP"-Befehl wieder den Befehl BRK, der einen Rücksprung in das Betriebsprogramm erzwingt. Soll das Programm als Unterprogramm verwendung finden, wird ein RTS-Befehl hierfür eingesetzt.

Das Programm im Maschinencode (hex):

```
0200  A5  LDA  ZP  01
0202  F0  BEQ  R   17
0204  4A  LSR  AC
0205  85  STA  ZP  03
0207  A5  LDA  ZP  02
0209  F0  BEQ  R   10
020B  4A  LSR  AC
```

```
020C C5  CMP  ZP  03
020E 10  BPL  R   08
0210 A4  LDY  ZP  01
0212 A6  LDX  ZP  02
0214 86  STX  ZP  01
0216 84  STY  ZP  02
0218 4C  JMP  AB  20  02
021B A9  LDA  IM  00
021D 85  STA  ZP  00
021F 00  BRK  IMP
0220 18  CLC  IMP
0221 A6  LDX  ZP  02
0223 A9  LDA  IM  00
0225 65  ADC  ZP  01
0227 C8  DEX  IMP
0228 D0  BNE  R   FB
022A 85  STA  ZP  00
022C 00  BRK  IMP
```

Der Speicherauszug:

```
0200 A5 01 F0 17 4A 85 03 A5 02 F0 10 4A C5 03 10 08
0210 A4 01 A6 02 86 01 84 02 4C 20 02 A9 00 85 00 00
0220 18 A6 02 A9 00 65 01 C8 D0 FB 85 00 00
```

6.2 Der Verschiebealgorithmus

Bevor wir uns dem schnelleren Algorithmus zuwenden, wollen wir für den ersten Algorithmus eine Rechenzeitabschätzung durchführen.

Soll kein Überlauf auftreten, so kann der ungünstigste Fall 15 x 15 = 225 sein. Das heißt, die CPU muß 15 Additionen (Schleifendurchläufe) durchführen.

Wie kann man dies nun abkürzen? Denken wir an unsere gewohnte Rechenmethode. Niemand addiert 15 mal, wenn er mit 15 multipli-

zieren will. Wir arbeiten nach einem Verschiebealgorithmus.

```
 15 x 15        und dual    1111 x 1111
 -------                    -----------
     75                            1111
    15                            1111
 -------
    225                          1111
                                1111
                            -----------
                             11100001   (die Überträge sind
                                         zu beachten!)
```

Wir führen die Multiplikation auf eine Verschiebeoperation mit anschließender Addition zurück. Im ungünstigsten Fall treten bei diesem Algorithmus nur drei Verschiebungen und Additionen auf. Man spart also einiges an Rechenzeit, was bei komplexen Multiplikationsprogrammen deutlich erkennbar wird.

Die Problemlösung:

Wie kann man nun diesen Algorithmus in ein geeignetes Programm übersetzen? Dazu vorher noch zwei Beispiele:

```
 1011 x 101                      1011 x 1000
 ----------                      -----------
       1011                        1011
      0000  kann entfallen       -----------
     1011                          1011000
 ----------                      nur drei Verschiebungen
     110111
```

zwei Verschiebungen
und eine Addition

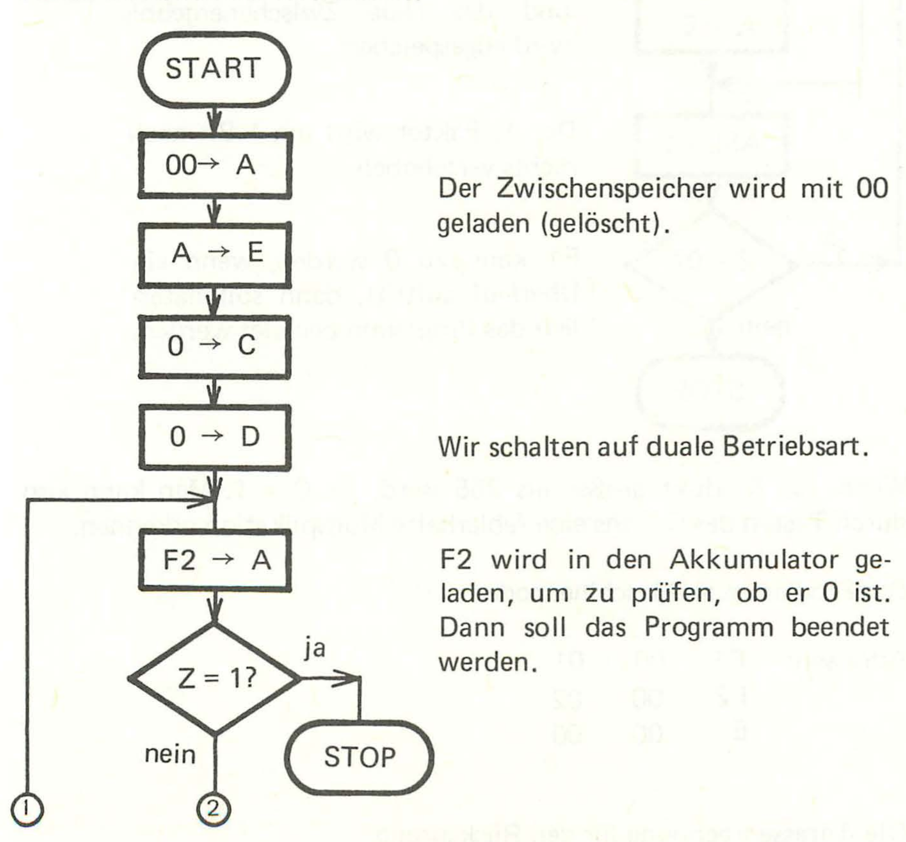
Eine kurze Diskussion soll folgen!
F1 sei der 1. Faktor, F2 der zweite.
Ist Bit 0 von F2 eins, dann wird F1 dazwischengespeichert. Anschließend wird F1 um eine Stelle nach links verschoben. Ist nun Bit 1 von

F2 eins, dann wird F1 zum Zwischenspeicher addiert, sonst unterbleibt dies. F1 wird wieder um eine Stelle nach links verschoben. Wenn Bit 2 von F2 eins ist, wird F1 zum Zwischenspeicher addiert, sonst unterbleibt dies und F1 wird wieder verschoben u.s.w.

Man erkennt, daß der Algorithmus in einer Schleife zu programmieren ist. Damit nicht übermäßig viele Durchläufe stattfinden, muß man noch eine geeignete Abbruchbedingung finden.
Im Zwischenspeicher steht nach Ablauf des Programmes das Produkt.

Nun die Diskussion des Ablaufplanes:

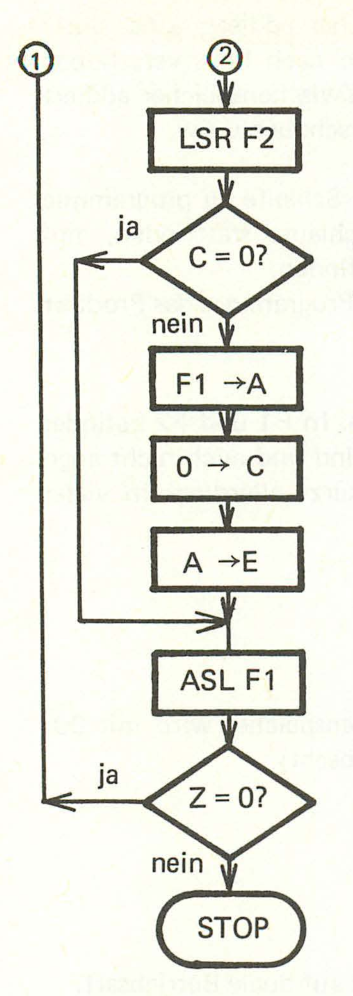
Der Zwischenspeicher soll E genannt werden. In F1 und F2 befinden sich die Faktoren, die nicht unbedingt $\neq 0$ sind und auch nicht angeordnet sein müssen. Eine Anordnung verkürzt allerdings in vielen Fällen erheblich die Rechenzeit.



Der Zwischenspeicher wird mit 00 geladen (gelöscht).

Wir schalten auf duale Betriebsart.

F2 wird in den Akkumulator geladen, um zu prüfen, ob er 0 ist. Dann soll das Programm beendet werden.



F2 wird um ein Bit nach rechts verschoben und Bit 0 kommt in das C-Flag.

War Bit 0 Null, so wird die folgende Addition übersprungen.

Der erste Faktor wird geladen.

der Zwischenspeicher hinzuaddiert

und das neue Zwischenergebnis wird abgespeichert.

Der 1. Faktor wird um 1 Bit nach rechts verschoben.

F1 kann zu 0 werden, wenn ein Überlauf auftritt, dann soll natürlich das Programm beendet werden.

Wenn das Produkt größer als 255 wird, ist C = 1. Man kann also durch Testen des C-Flags eine fehlerhafte Multiplikation erkennen.

Das Programm im Maschinencode:

Adressen:	F1	00	01
	F2	00	02
	E	00	00

Die Adressenrechnung für den Rücksprung:

20 Bytes werden übersprungen

20	=	0001 0100	(dual)
		1110 1011	(Komplement)
		+ 1	
EC_{16}	=	1110 1100	(2er-Komplement)

Da dieses Programm ebenfalls beliebig im Speicher verschiebbar ist, kann man das Sortierprogramm des vorherigen Beispiels auch vor dieses Programm setzen.

Programm im Maschinencode:

```
0200  a9  lda im  00
0202  85  sta zp  00
0204  18  clc imp
0205  d8  cld imp
0206  a5  lda zp  02
0208  d0  bne r   01
020a  00  brk imp
020b  46  lsr zp  02
020d  90  bcc r   07
020f  a5  lda zp  01
0211  18  clc imp
0212  65  adc zp  00
0214  85  sta zp  00
0216  06  asl zp  01
0218  d0  bne r   ec
021a  00  brk imp
```

Der Speicherauszug:

```
0200 a9 00 85 00 18 d8 a5 02 d0 01 00 46 02 90 07 a5
0210 01 18 65 00 85 00 06 01 d0 ec 00
```

7. Die Division

Auch zur Division sollen zwei verschiedene Algorithmen vorgestellt werden.

Da die Division als inverse (gegensätzliche) Operation der Multiplikation erklärt werden kann, ist leicht einsehbar, daß man sie auf eine Mehrfachsubtraktion zurückführen kann.

Das bekannte Verfahren der „schriftlichen Division" wird im zweiten Programmbeispiel vorgestellt.

Leider ist bei der Division eine Vertauschung von Dividenden und Divisor nicht möglich. Daher sind im Schnitt beide Rechenprogramme langsamer.

7.1 Die Subtraktionsmethode

25 : 7 = 3 Rest 4

Wir können von 25 die 7 dreimal subtrahieren.

25 – 7 – 7 – 7 = 4

Der Subtraktionsvorgang wird abgebrochen, wenn die Differenz kleiner als 7 ist.

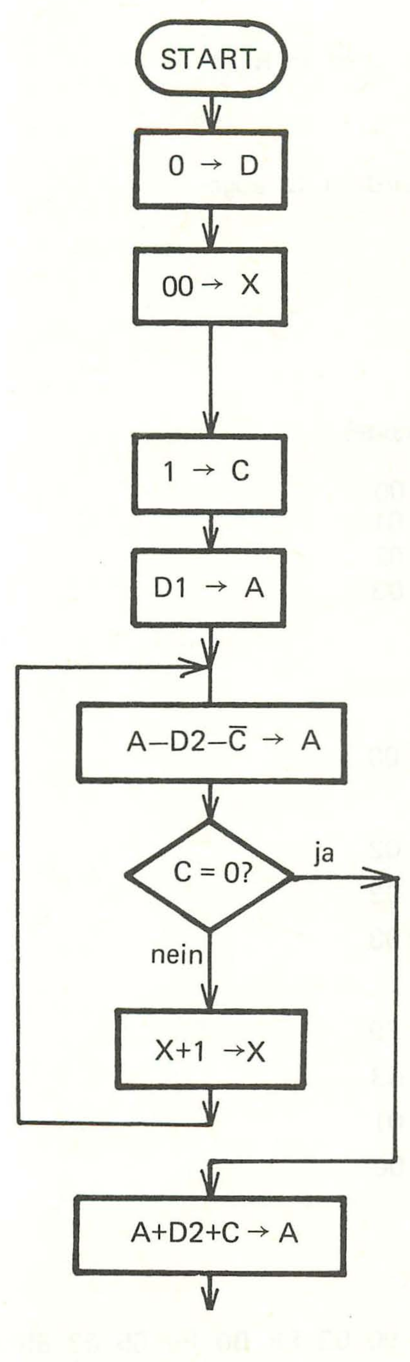
Bevor wir mit einer Division beginnen, müssen wir natürlich darauf achten, daß der Divisor (hier die 7) nicht 0 ist. Das wäre eine unerlaubte Division und würde bei unserem Verfahren zu einer Endlosschleife führen. Es tritt dann nie die Abbruchbedingung ein.

Der Lösungsweg:

Die Darstellung des Subtraktionsalgorithmus in einem Ablaufplan wird uns nicht mehr schwerfallen.

Wir legen fest:

	D1	ist der Dividend, der auch 0 sein darf,
	D2	ist der Divisor, der immer ungleich 0 sein muß.
	Q	sei der Quotient.
	R	sei der Rest.



duale Betriebsart

Wir verwenden das X-Register als Zähler der Subtraktionen. Der Quotient muß vor der Division natürlich 0 sein.

C muß vor eine Subtraktion gesetzt werden.

Der Divident wird geladen.

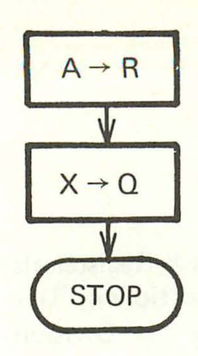
Der Divisor wird subtrahiert.

Ergab die Differenz einen Überlauf? Wenn ja, muß die Schleife beendet werden. Es soll nun der Divisionsrest bestimmt werden.

X zählt weiter.

Nun beinhaltet X den Quotienten.

Um den Rest zu bestimmen, wird D2 noch einmal addiert.



Der Divisionsrest wird in R abgelegt.

Der Quotient wird in Q abgespeichert.

Das Programm im Maschinencode (Hexcode):

Wir wählen als Adressen:

Q	00 00
R	00 01
D1	00 02
D2	00 03

Adresse	Code	Befehl	Modus	Operand
02 00	D8	CLD	IMP	
01	A2	LDX	IM	00
03	38	SEC	IMP	
04	A5	LDA	ZP	02
06	E5	SBC	ZP	03
08	90	BCC	R	03
0A	E8	INX	IMP	
0B	B0	BCS	R	F9
0D	65	ADC	ZP	03
0F	85	STA	ZP	01
11	86	STX	ZP	00
13	00	BRK	IMP	

Der Speicherauszug:

0200 D8 A2 00 38 A5 02 E5 03 90 03 E8 B0 F9 65 03 85
0210 01 86 00 00

Die Adressenrechnung für den Rücksprung:

Es werden 7 Bytes übersprungen.

7	=	0000 0111	(dual)
		1111 1000	(Komplement)
		+1	
$F9_{16}$	=	1111 1001	(2er-Komplement)

Abschließend noch eine Abschätzung:
Der ungünstigste Divisionsfall ist 255 : 1. Dann müssen 256 Subtraktionen durchgeführt werden. Das bedeutet bei höherer Rechengenauigkeit eine beträchtliche Rechenzeit. Daher wird der Programmierer nach einem schnelleren Verfahren suchen.

7.2 Der Verschiebealgorithmus

Dieser Algorithmus ist wesentlich schneller als die Subtraktionsmethode (im Schnitt um den Faktor 16). Er erfordert aber auch einige Programmierarbeit.

Wie sind wir gewohnt, die Division 255 : 5 zu lösen?

```
Doch so: 255 : 5 = 51
        -25
         05
        -05
         00
```

```
Im Dualsystem:   Dividend       Divisor     Quotient
                 1111 1111 :    101      =  11 0011
               - 101
                   101
                 -101
                     0 111
                      -101
                        101
                       -101
                        000
```

ausführlicher geschrieben:

```
  1111 1111 : 101 = 110011
- 1010 0000
-----------
   101 1111
-  101 0000
-----------
   000 1111
   -   1010
-----------
       0101
      - 101
-----------
        000
```

Wir wollen nun versuchen, diesen Algorithmus in Worte zu fassen.

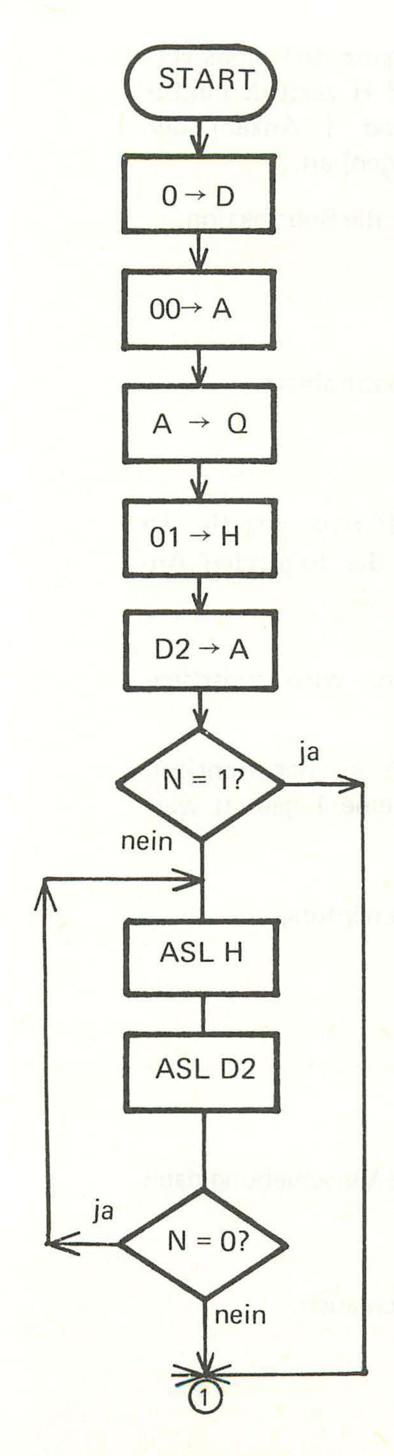
Der Divisor wird zu Beginn der Division ganz nach links verschoben. Dann wird er vom Dividenden subtrahiert. Ist die Differenz positiv oder Null, so wird eine 1 gesetzt und die Differenz wird zwischengespeichert. Ist die Differenz negativ, so geschieht dies nicht und es wird eine Null gesetzt.

Wir verschieben den Divisor nun um eine Stelle nach rechts und subtrahieren ihn vom Zwischenergebnis. Ist die Differenz positiv oder Null, so wird eine 1 gesetzt und die neue Differenz wird zwischengespeichert. Ist sie negativ, so geschieht dies nicht und es wird eine 0 gesetzt. Es folgt wieder eine Rechtsverschiebung des Divisors. Es läuft immer wieder das gleiche ab, bis der Divisor in seine alte Position zurückgelangt ist. Dann wird die Division abgebrochen.

Wir müssen also mehrfach eine Rechtsverschiebung mit anschließender Subtraktion ausführen und uns genau die aktuelle Stellung des Divisors merken.

Dem Leser sei empfohlen, noch einige Divisionen zu üben, bevor er sich der Diskussion des folgenden Ablaufplanes zuwendet.

Die Problemlösung:
Wir legen fest:
D1 sei der Dividend, D2 der Divisor, der nicht 0 sein darf. Q ist der Quotient und H ein Hilfsregister, das den aktuellen Stand der Verschiebungen von D2 protokolliert und das „Setzen" der 1 durchführt.



Duale Betriebsart

Alle Bits des Quotienten werden gelöscht.

D2 befindet sich ja im „Ausgangszustand. Daher wird die 1 in H ganz rechts angeordnet.

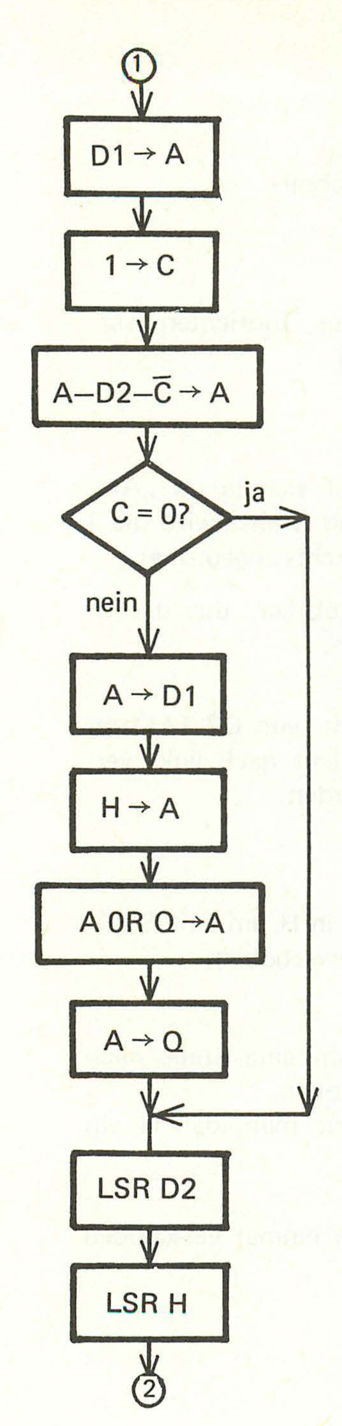
D2 wird geladen, um das 8. Bit zu testen.

Ist das 8. Bit von D2 1? Dann kann D2 nicht nach links verschoben werden.

Die 1 wird in H um eine Stelle nach links verschoben.

D2 wird um eine Stelle nach links verschoben.
Hier erkennt man, daß H ein Zeiger ist.

Kann noch einmal verschoben werden?



D2 ist nun ganz nach links verschoben und H zeigt den aktuellen Zustand (Anzahl der Verschiebungen) an.

Nun beginnt die Subtraktion.

Es wird D2 subtrahiert.

Ist die Differenz negativ, so überspringe die folgenden Anweisungen.

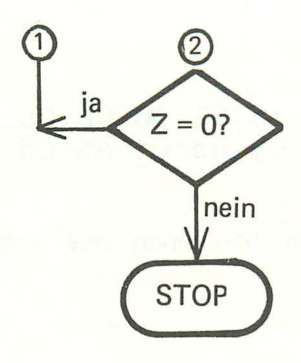
Die Differenz wird zwischengespeichert.

Es soll nun an der richtigen Stelle in Q eine 1 gesetzt werden.

Die Oderverknüpfung.

Es folgt eine Verschiebung nach rechts.

In H natürlich auch.



Ist H noch nicht 0, so folgt eine weitere Subtraktion. Ist H=0, so befindet sich D2 wieder in der ursprünglichen Position und die Division ist beendet.

Das Programm im Maschinencode (Hexcode):

Die Adressen:	Q	00	00
	D1	00	01
	D2	00	02
	H	00	03

```
0200  d8  cld imp
0201  a9  lda im  00
0203  85  sta zp  00
0205  a9  lda im  01
0207  85  sta zp  03
0209  a5  lda zp  02
020b  30  bmi r   06
020d  06  asl zp  03
020f  06  asl zp  02
0211  10  bpl r   fa
0213  a5  lda zp  01
0215  38  sec imp
0216  e5  sbc zp  02
0218  90  bcc r   08
021a  85  sta zp  01
021c  a5  lda zp  03
021e  05  ora zp  00
0220  85  sta zp  00
0222  46  lsr zp  02
0224  46  lsr zp  03
0226  d0  bne r   eb
0228  00  brk imp
```

Der Speicherauszug:

```
0200 d8 a9 00 85 00 a9 01 85 03 a5 02 30 06 06 03 06
0210 02 10 fa a5 01 38 e5 02 90 08 85 01 a5 03 05 00
0220 85 00 46 02 46 03 d0 eb 00
```

Es kommen zwei Rücksprünge vor. Wir berechnen zwei relative Adressen.

1. Es werden 6 Bytes übersprungen.

	0000 0110	(dual)
	1111 1001	
	+ 1	
FA_{16} =	1111 1010	(2er-Komplement)

2. Es werden 21 Bytes übersprungen.

	0001 0101	
	1110 1010	
	+ 1	
EB_{16} =	1110 1011	(2er-Komplement)

Ein Probelauf:

Wurde das Programm fehlerfrei geladen, so wollen wir in die Speicherzelle 00 01 die Zahl $25_{10} = 19_{16}$ und die Speicherzelle 00 02 die Zahl $5_{10} = 05_{16}$ eingeschrieben.

Dann wird das Programm ab der Stelle 0200 gestartet. Das Byte D2 wird jetzt 5 mal nach links verschoben. Am Ende der Verschiebeschleife befindet sich in D2 das Byte 1010 0000 = $A0_{16}$ und in H das Byte 0010 0000 = 20_{16}.

Nun beginnt die erste Subtraktion: $19_{16} - A0_{16}$. Die Differenz ist kleiner als Null, D2 und H werden um eine Stelle nach rechts verschoben.

D2 = $01010000_2 = 50_{16}$
H = $0001\,0000_2 = 10_{16}$

Da H ungleich Null ist, folgt die nächste Subtraktion: $19_{16} - 50_{16}$. Die Differenz ist wieder kleiner als Null. D2 und H werden erneut um eine Stelle nach rechts verschoben.

$D2 = 0010\,1000_2 = 28_{16}$, $H = 0000\,1000_2 = 08_{16}$

Es folgt die 3. Subtraktion: $19_{16} - 28_{16}$. Wieder ist die Differenz kleiner als Null. Eine weitere Rechtsverschiebung von D2 und H erfolgt. Nun:

$D2 = 0001\,0100_2 = 14_{16}$, $H = 0000\,0100_2 = 04_{16}$

Die Differenz ist jetzt $19_{16} - 14_{16} = 05_{16}$. Sie wird in D1 zwischengespeichert. Die Oderverknüpfung setzt in Q das 3. Bit: Q = 0000 0100. Anschließend werden D2 und H um eine Stelle nach rechts verschoben:

$D2 = 0000\,1010_2 = 0A_{16}$, $H = 0000\,0010_2 = 02_{16}$.

Die 5. Differenz ist kleiner als Null: $05_{16} - 0A_{16}$. D2 und H werden noch einmal nach rechts verschoben:

$D2 = 0000\,0101_2 = 05_{16}$, $H = 0000\,0001_2 = 01_{16}$.

Es folgt die 6. Subtraktion mit der Differenz Null. Die anschließende Oderverknüpfung setzt das 1. Bit in Q, so ergibt sich: $Q = 0000\,0101_2 = 05_{16}$. Bei der anschließenden Rechtsverschiebung von D2 und H wird Z = 1 und damit der Programmablauf beendet.

8. Umrechnung von Zahlen im Dezimalsystem in das Dualsystem

Die Unterprogrammtechnik:

In diesem Abschnitt verwenden wir zum erstenmal die Unterprogrammtechnik. Dies ist eine sehr hilfreiche Programmiertechnik. Sie vereinfacht die Programmierarbeit wesentlich. Drei Gründe sprechen für eine häufige Verwendung dieser Technik.

1. Wenn gleiche Abläufe in einem Programm mehrmals vorkommen, so ist es platzsparender, sie nur ein einziges Mal zu programmieren und in ein Programm einzufügen. Als Unterprogramm angelegt können diese Abläufe von beliebigen Stellen im Hauptprogramm aufgerufen werden.

2. Auch wenn ein Ablauf in einem bestimmten Programm nur einmal Verwendung findet, ist es oft sinnvoll, ihn als Unterprogramm zu gestalten. Dies geschieht vor allem, um ein Programm übersichtlich zu gestalten. Die Programmdokumentation ist damit klarer.

3. Bestimmte, wichtige Abläufe irgendeines Programmes sind als Unterprogramme universeller verwendbar. Man kann sie in einer <u>Unterprogrammbibliothek</u> anlegen und damit für andere Programme greifbar gestalten. Später zu entwickelnde Programme können dann auf die Unterprogramme der Bibliothek zurückgreifen.

Bei der Anlage von Unterprogrammen muß die Dokumentation sorgfältig gestaltet werden. Aus der Dokumentation muß deutlich erkennbar sein:

1. welche Aufgabe es hat,
2. welche CPU-Register und Speicherzellen beeinflußt werden,
3. auf welche Weise die Unterprogrammparameter (Daten, Indizes etc.) an das Unterprogramm zu übergeben sind,
4. auf welche Weise die Ergebnisse der Unterprogrammoperationen an das Hauptprogramm zurückgegeben werden.

Soll die CPU ein Hauptprogramm an einer bestimmten Stelle verlassen, um ein Unterprogramm abzuarbeiten, so ist dafür Sorge zu tragen, daß sie wieder an die richtige Stelle im Hauptprogramm nach der Beendigung des Unterprogramms zurückfindet. Es muß der aktuelle Inhalt des PC (Programmzählers) in einem bestimmten Speicherbereich zwischengespeichert werden, um nach der Unterprogrammdurchführung in den PC zurückgeladen zu werden. Besitzt ein Microprozessor hierfür keine speziellen Unterprogrammbefehle, so muß dies ein spezieller Befehlssatz im Hauptprogramm und am Ende des Unterprogramms durchführen. Als Adresszwischenspeicher verwendet man dann meist die ersten beiden Bytes zu Beginn des Unterprogramms.

Der 6502 besitzt nun spezielle Unterprogrammbefehle (s.Kap. 3.3). Der Befehl JSR befindet sich im Hauptprogramm und erzwingt einen Sprung an die angegebene Anfangsadresse des betreffenden Unterprogramms. Die Adresse des Befehls, der auf den JSR-Befehl folgt, wird dabei in den Kellerspeicher (Stack) abgelegt. Der Befehl RTS, der ein Unterprogramm beendet, bewirkt ein Zurückladen der letzten, im Stack abgespeicherten Adresse in den PC. Damit hat die CPU an der richtigen Stelle in das Hauptprogramm zurückgefunden.

Man kann bis zu 128 Unterprogramme ineinander verschachteln, da der Stackbereich eine RAM-Seite umfaßt. Es werden die Rücksprungadressen automatisch in der richtigen Reihenfolge angeordnet. Dies ist natürlich keine eigentliche Beschränkung, da man mit geeigneten Befehlsfolgen die Adressen auch an anderen Speicherstellen ablegen kann.

Da in diesem Beispiel einige Abläufe sich mehrfach wiederholen, benutzen wir Unterprogramme. Aus dem 2. und 3. Grunde werden große und komplexe Programmsysteme (z. B. Betriebssysteme) in viele Unterprogramme zerlegt.

Die Problemstellung

Es soll eine Dezimalzahl in das duale Zahlensystem umgerechnet (konvertiert) werden. Der Zahlenbereich umfaßt nur die Zahlen von 0 bis 255, da wir uns auf 1 byte Zahlen (in dualer Darstellung) beschränken wollen.

Die Problemanalyse:

Einige mathematische Grundlagen zur Zahlenkonvertierung kann man in Kapitel 2.1 finden.

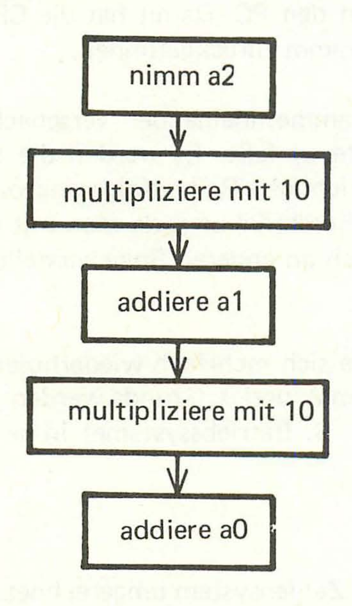
Wenn a0, a1 und a2 die Ziffern einer Zahl Z im Dezimalsystem sind, läßt sich diese bekanntlich durch

$Z = a2 \times 100 + a1 \times 10 + a0$ darstellen.

Dies kann man umformen:

$Z = (a2 \times 10 + a1) \times 10 + a0.$

Hier steht nun schon die Umrechnungsvorschrift!
Liegen die Ziffern schon in dualer Form vor (BCD-System), muß man nur mehrfach mit 10 (natürlich dual) multiplizieren und anschließend die nächste Ziffer addieren. Folgender grober Ablaufplan wäre denkbar.



Würden wir für jede Dezimalziffer eine getrennte Speicherzelle verwenden, könnte man diesen Plan gleich in ein Programm umsetzen.

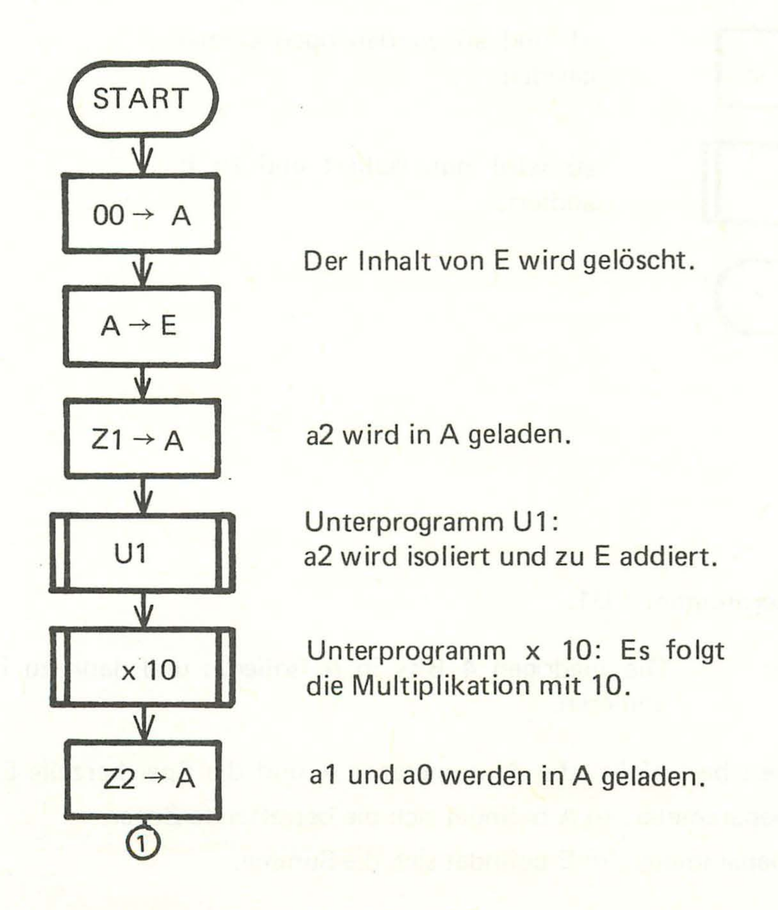
Die Ziffern sollen aber in gepackter Form (zwei Ziffern pro Byte) eingegeben werden. Daher muß noch für eine geeignete Trennung der Ziffern gesorgt werden.

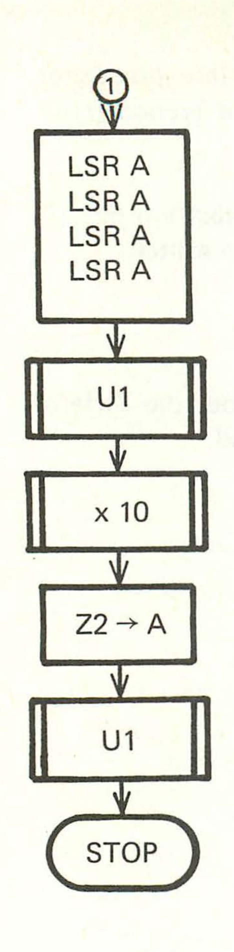
Den groben Plan entnehmen wir auch, daß die Multiplikation mit 10 und die Addition sinnvoll in Unterprogrammen ablaufen sollten.

Die Problemlösung:

Wir vereinbaren:

- E sei das Ergebnis
- Z1 sei die höchste Ziffer a2,
- Z2 sei die Speicherstelle, in der die Ziffern a1 und a0 abgespeichert sind.





a0 wird herausgeschoben.

Unterprogramm U1: a1 wird isoliert und zu E addiert.

Die zweite Multiplikation mit 10.

a1 und a0 werden noch einmal geladen.

a0 wird nun isoliert und zu E addiert.

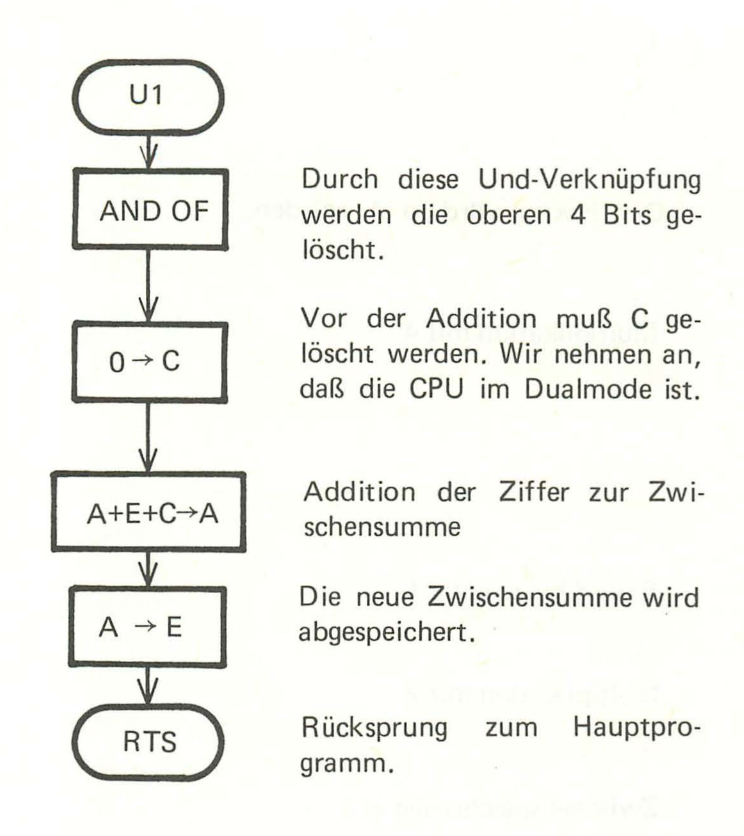
Das Unterprogramm U1:

Aufgabe: Die niedrigen 4 Bits in A isolieren und dann zu E addieren.

Es werden beeinflußt: der Akkumulator A und die Speicherzelle E.

Übergabeparameter: In A befindet sich die betreffende Ziffer.

Rückgabeparameter: In E befindet sich die Summe.



Das Unterprogramm x 10

Da eine Multiplikation mit 2 einer Linksverschiebung entspricht, ist es sinnvoll (Einsparung von Rechenzeit), die Multiplikation mit 10 auf möglichst viele Verdopplungen zu reduzieren.

Da $10 = 8 + 2 = 2 \times 2 \times 2 + 2$
gilt: $a \times 10 = ((a \times 2) \times 2 + a) \times 2$. Wobei a ein beliebiger Faktor ist.

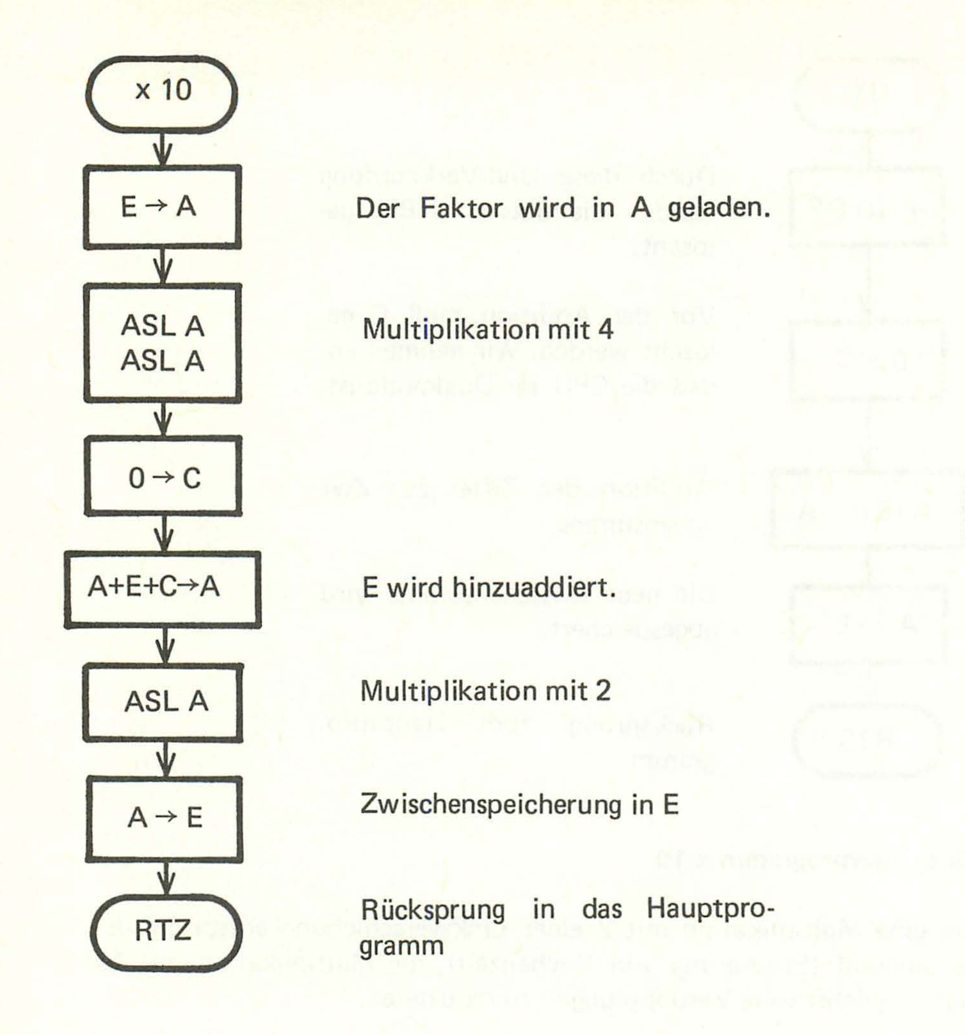
Die Rechenvorschrift kann nun leicht programmiert werden.

Aufgabe: Multiplikation von E mit 10

Es werden beeinflußt: der Akkumulator A und die Zelle E.

Übergabeparameter: in E befindet sich der Faktor.

Rückgabeparameter: In A und E befindet sich das Produkt.



Das Programm im Maschinencode:

Adressen:	E	00	00
	Z1	00	01
	Z2	00	02

Das Hauptprogramm ist von Adresse 0200 bis Adresse 021F angelegt. An seinem Ende wurden zwei NOP-Befehle eingefügt, um das nachträgliche Anbringen eines 3 byte-Sprungbefehls zu ermöglichen, falls dies erwünscht ist.

U1 liegt im Bereich 0220 bis 0227 und x 10 beansprucht den restlichen Speicherbereich des Programmes.

```
0200  a9  lda im  00
0202  85  sta zp  00
0204  a5  lda zp  01
0206  20  jsr abs 20  02
0209  20  jsr abs 28  02
020c  a5  lda zp  02
020e  4a  lsr ac
020f  4a  lsr ac
0210  4a  lsr ac
0211  4a  lsr ac
0212  20  jsr abs 20  02
0215  20  jsr abs 28  02
0218  a5  lda zp  02
021a  20  jsr abs 20  02
021d  00  brk imp
021e  ea  nop imp
021f  ea  nop imp
0220  29  and im  0f
0222  18  clc imp
0223  65  adc zp  00
0225  85  sta zp  00
0227  60  rts imp
0228  a5  lda zp  00
022a  0a  asl ac
022b  0a  asl ac
022c  18  clc imp
022d  65  adc zp  00
022f  0a  asl ac
0230  85  sta zp  00
0232  60  rts imp
```

Der Speicherauszug:

```
0200 a9 00 85 00 a5 01 20 20 02 20 28 02 a5 02 4a 4a
0210 4a 4a 20 20 02 20 28 02 a5 02 20 20 02 00 ea ea
0220 29 0f 18 65 00 85 00 60 a5 00 0a 0a 18 65 00 0a
0230 85 00 60
```

9. Umrechnung von Zahlen im Dualsystem in das Dezimalsystem

Die Problemstellung:

Es soll eine 8 bit Dualzahl in eine maximal dreistellige Dezimalzahl umgerechnet werden.

Die Problemanalyse:

Dies ist nun ein Beispiel für eine Umrechnung einer Zahl von einem Zahlensystem mit einer kleineren Basis (Basis 2) in ein Zahlensystem mit einer größeren Basis (Basis 10).
Die Umrechnung kann nur mit dem etwas aufwendigen Divisionsalgorithmus (siehe Kapitel 2.1) durchgeführt werden.

Z sei die Zahl in dualer Darstellung. In dezimaler Darstellung gilt:
Z = a2 x 100 + a1 x 10 + a0

a2, a1 und a0 sind die Dezimalziffern, die berechnet werden. Diese können wir durch Division mit Rest bestimmen.
Man berechnet zuerst die höchste Ziffer. Dann nimmt man den Divisionsrest und bestimmt die nächst niedere Ziffer.

Z = a2 x 100 + Z1 (Z1 ist der 1. Rest)
Z1 = a1 x 10 + a0 (a0 ist der 2. Rest)

Die Problemlösung:

Wir erkennen, daß zwei Divisionen durchgeführt werden, bei denen der Divisionsrest für den nächsten Rechenschritt weitere Verwendung findet. Es erscheint daher sinnvoll, ein Divisionsunterprogramm zu entwickeln, das wir zuerst betrachten.

Das Divisionsunterprogramm

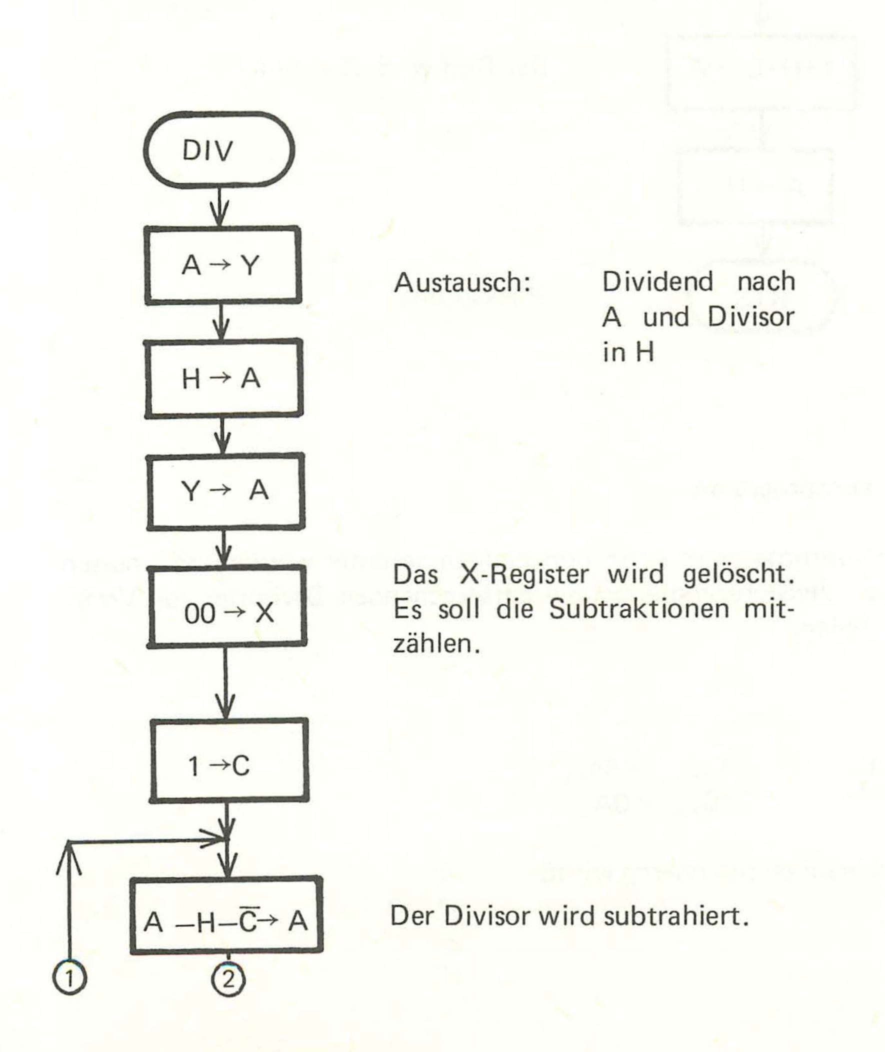
Es soll der Subtraktionsalgorithmus benutzt werden. Als Eingabeparameter müssen der Dividend und der Divisor zur Verfügung stehen. Als Ausgabeparameter sollen der Quotient und der Divisionsrest übergeben werden.

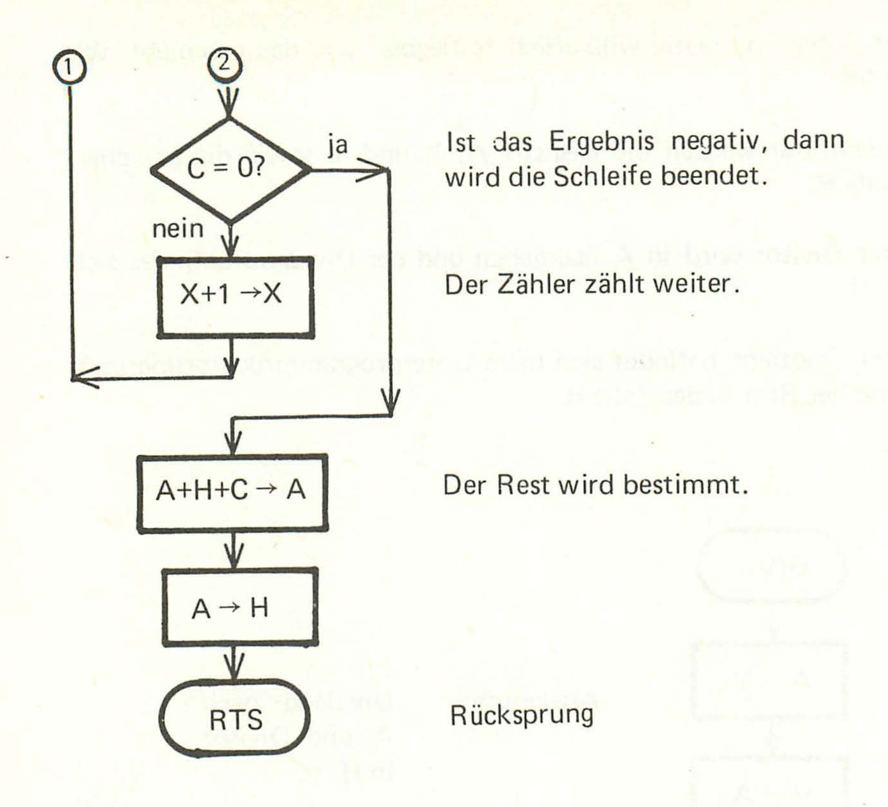
Es ist natürlich relativ willkürlich festlegbar, wie das geschieht. Wir legen fest:

1. Beeinflußt werden die Register A, X und Y sowie die Speicherzelle H.

2. Der Divisor wird in A übergeben und der Dividend befindet sich in H.

3. Der Quotient befindet sich beim Unterprogrammrücksprung in X und der Rest in der Zelle H.



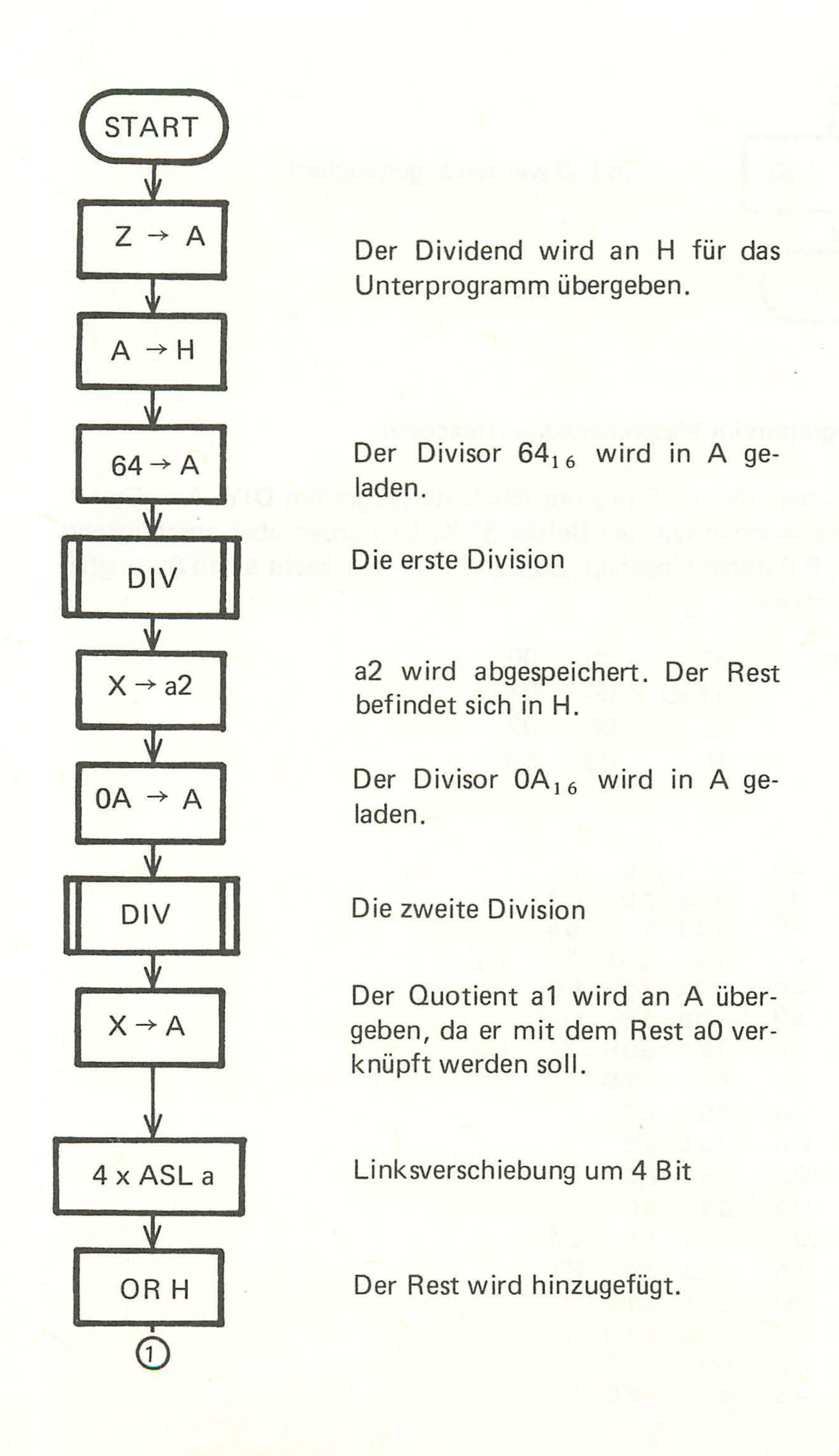


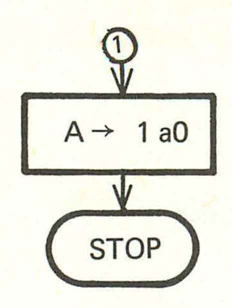
Das Hauptprogramm:

Das Hauptprogramm kann nun einfach gestaltet werden. Wir müssen für das Divisionsprogramm die entsprechenden Divisoren zur Verfügung stellen.

Es gilt: $100_{10} = 64_{16}$
$10_{10} = 0A_{16}$

Speicherzellen reservieren wir für: a2
a1 a0
Z
H


START
Z → A
A → H
64 → A
DIV
X → a2
0A → A
DIV
X → A
4 x ASL a
OR H
1
Der Dividend wird an H für das Unterprogramm übergeben.
Der Divisor 64_{16} wird in A geladen.
Die erste Division
a2 wird abgespeichert. Der Rest befindet sich in H.
Der Divisor $0A_{16}$ wird in A geladen.
Die zweite Division
Der Quotient a1 wird an A übergeben, da er mit dem Rest a0 verknüpft werden soll.
Linksverschiebung um 4 Bit
Der Rest wird hinzugefügt.



a1 und a0 werden abgespeichert.

Das Programm im Maschinencode (Hexcode):

Ab Speicherzelle 021C beginnt das Unterprogramm DIV. Als „Stop"-Befehl verwenden wir den Befehl BRK. Es werden aber anschließend zwei NOP-Befehle eingefügt. So kann man hier leicht einen Sprungbefehl einsetzen.

Adressen:			
	a2	00	00
	a1 a0	00	01
	Z	00	02
	H	00	03

```
0200   a5   lda zp   02
0202   85   sta zp   03
0204   a9   lda im   64
0206   20   jsr abs  1c   02
0209   86   stx zp   00
020b   a9   lda im   0a
020d   20   jsr abs  1c   02
0210   8a   txa imp
0211   0a   asl ac
0212   0a   asl ac
0213   0a   asl ac
0214   0a   asl ac
0215   05   ora zp   03
0217   85   sta zp   01
0219   00   brk imp
021a   ea   nop imp
021b   ea   nop imp
021c   a8   tay imp
```

```
021d  a5  lda zp  03
021f  84  sty zp  03
0221  a2  ldx im  00
0223  38  sec imp
0224  e5  sbc zp  03
0226  90  bcc r   04
0228  e8  inx imp
0229  4c  jmp abs 24  02
022c  65  adc zp  03
022e  85  sta zp  03
0230  60  rts imp
```

Der Speicherauszug:

```
0200  a5 02 85 03 a9 64 20 1c 02 86 00 a9 0a 20 1c 02
0210  8a 0a 0a 0a 0a 05 03 85 01 00 ea ea a8 a5 03 84
0220  03 a2 00 38 e5 03 90 04 e8 4c 24 02 65 03 85 03
0230  60
```

4.1.2 Zwei Grundprogramme für Steuerungsanlagen

Die Microcomputer haben schon ihre Bedeutung in der Steuerungstechnik bewiesen. Viele frei programmierbare Steuerungsanlagen besitzen als Herz einen Microcomputer. Die Bedeutung der Microcomputer in der Steuerungstechnik wird in der nächsten Zeit noch wesentlich wachsen. Sie werden zunehmend die digitalen Netzwerke und die mechanischen Steuerungen ersetzen.
Digitale Netzwerke haben in der Steuerungstechnik folgende Aufgaben:

1. Steuerung periodischer Abläufe. Sie erzeugen also periodische Rechtecksignale auf mehreren verschiedenen Kanälen, die dann die verschiedenen Stellglieder schalten.

2. Die Steuerungsdaten (numerische Daten) befinden sich auf externen Datenträgern (Lochstreifen, Magnetband). Die zeitliche Abfolge der Abarbeitung und die Umsetzung in den für die Maschinen zugeschnittenen Code übernimmt ein digitales Netzwerk.

3. Digitales Regelglied: Es besteht eine Rückkopplung vom Arbeitsprozess. Es werden bestimmte „Ist-Werte" verglichen mit „Soll-Werten" und daraus eine Stellgröße berechnet.

Das Grundelement einer Steuerungsanlage, die für den ersten Aufgabenkreis konstruiert wurde, ist ein Impulsgenerator. Im ersten Beispiel wollen wir lernen, wie man einen Microcomputer als Rechteckgenerator (Impulsgenerator) programmiert.

Ein solcher Generator sollte leicht auf beliebige Schaltzeiten umprogrammierbar sein. Wie man einen solchen Generator programmiert, zeigt das zweite Programmbeispiel.

In Kapitel 4.4 wird ein mehrstelliger Generator vorgestellt, der das Herz einer Ampelanlage werden könnte. Das zweite Beispiel von Kapitel 4.4 ist eine nicht triviale Ampelsteuerung. Digitale Netzwerke, die für den zweiten Aufgabenkreis konstruiert wurden, sind Codewandler. Auch zu Codewandlern sind Microcomputer leicht program-

mierbar. Entsprechende Beispiele stellt das Kapitel 4.1.3 vor.

Der wesentliche Vorzug eines Microcomputers liegt allerdings in der relativ einfachen Realisierung einer Steuerungsanlage mit Rückkopplung zum Arbeitsprozess (Regelprozess). Der Umfang dieses Buches würde gesprengt, wenn auch hierfür ein Beispiel vorgestellt wird, da der notwendige Aufwand an Hardware zwangsläufig größer ist. Man müßte geeignete digital-analog-Wandler und analog-digital-Wandler entwickeln.

1. Ein Rechteckgenerator

Die Problemstellung:

Ein Ausgangskanal unseres Microcomputers soll periodisch die Zustände 5V (1) und 0V (0) einnehmen.

Die Problemanalyse:

Die meisten Microcomputer besitzen Interfacebausteine, die die Verbindung zwischen dem Systembus und der Anwenderperipherie herstellen. Der KIM besitzt zum Beispiel zwei solche Bausteine. Diese Bausteine bieten meist einen 8 bit - parallel - Kanal an. Das Bit 0 eines solchen Ausgangskanals wollen wir als unseren Ausgangskanal ansehen. Die Interfacebausteine des KIM sind in den ICs 6530 integriert. Sie sind als Eingabeinterface oder als Ausgabeinterface programmierbar. Dies kann sogar bitweise geschehen.

Ein solches Interface besteht daher aus zwei Registern, dem Datenrichtungsregister, das die acht Kanäle programmiert und dem Datenregister, das die Daten auf die Kanäle abgibt oder von ihnen empfängt.

Die CPU komuniziert mit den Interfacebausteinen auf die gleiche Art wie mit einer RAM-Zelle. Will sie Daten auf die Ausgangskanäle geben, führt sie einen RAM-Schreibbefehl aus. Sollen von Eingabekanälen Daten empfangen werden, geschieht das durch einen RAM-Lesebefehl. Natürlich muß der Programmierer dafür sorgen, daß das Interface entsprechend programmiert wurde. Dies geschieht durch Ein-

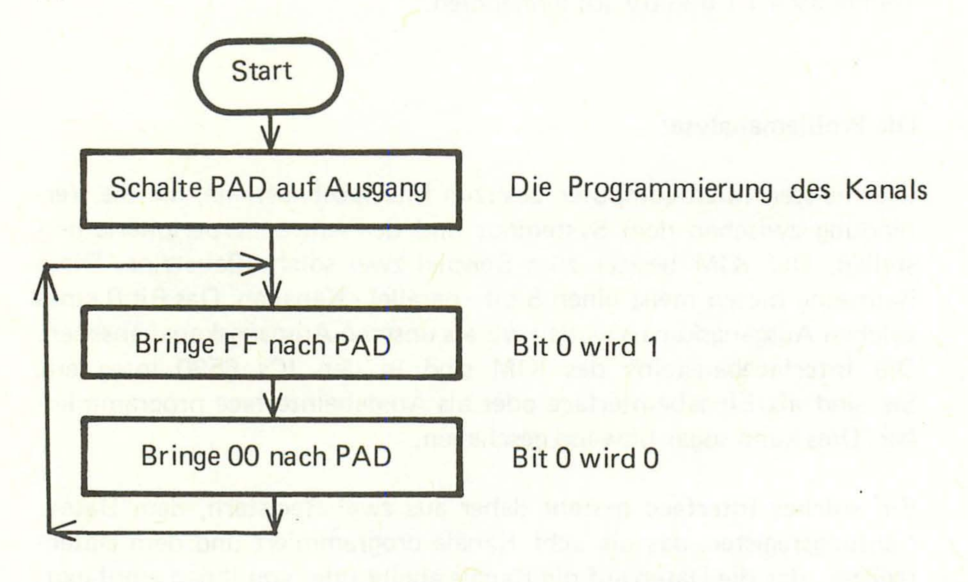
schreiben eines Steuerwortes in das zugehörige Datenrichtungsregister des Interfaces.

Beim 6530 wird durch Einschreiben einer 1 in das entsprechende Bit des Datenrichtungsregisters der Interfacekanal auf Ausgang geschaltet. Eine 0 bedeutet Eingang.

Die Problemlösung:

Wir wählen als Ausgangskanal das Bit 0 des Port A aus. Das Datenregister (PAD) hat die Adresse 1700, das Datenrichtungsregister (PADD) besitzt die Adresse 1701.

Da periodisch immer wiederkehrend die gleichen Operationen durchzuführen sind, befindet sich die CPU in einer Endlosprogrammschleife.



Dieser Generator arbeitet mit einer sehr hohen Frequenz.

Das Programm im Assemblercode:

Die Vorbereitung:	LDA	IM	FF	
	STA	AB	01	17

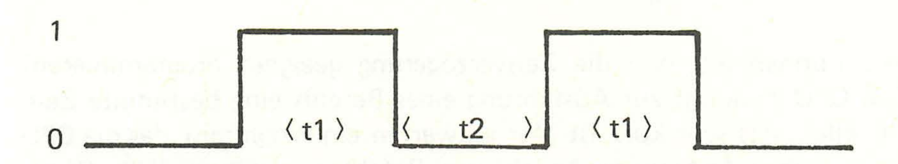
Bit 0 auf 1:	M	LDA	IM	FF	
		STA	AB	00	17
Bit 0 auf 0:		LDA	IM	00	
		STA	AB	00	17
Die Schleife wird geschlossen:		JMP	AB	M	

Die Übersetzung des Programmes in den Maschinencode wird dem Leser zur Übung überlassen.

2. Ein programmierbarer Rechtecksgenerator

Die Problemstellung:

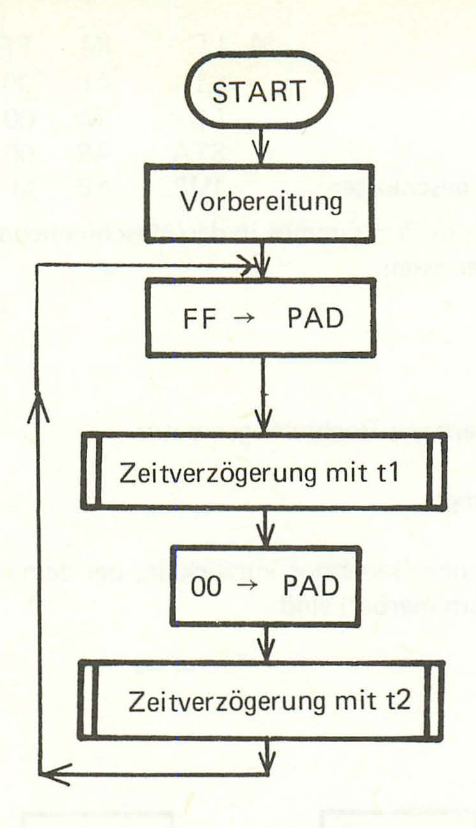
Wir wollen nun einen Generator entwickeln, bei dem die Schaltzeiten einstellbar (programmierbar) sind.



Die Schaltzeiten sind Vielfache von 1 ms.

Die Problemlösung:

Wir erweitern das eben besprochene Programm geeignet.



Wie können wir nun die Zeitverzögerung geeignet programmieren? Die CPU benötigt zur Ausführung eines Befehls eine bestimmte Zeit, die allerdings sehr kurz ist. Wir entwerfen ein Programm, das die CPU eine genau festgelegte Anzahl von Befehlen ausführen läßt. Dieses Programm wirkt sich wie eine Verzögerung mit einer bestimmten Zeitkonstanten aus.

Ein einfaches Programm, das eine kurze Zeitverzögerung bewirkt, ist die folgende Programmschleife.

```
      LDX   IM    Zeitkonstante
M     DEX   IMP
      BNE   R     M
```

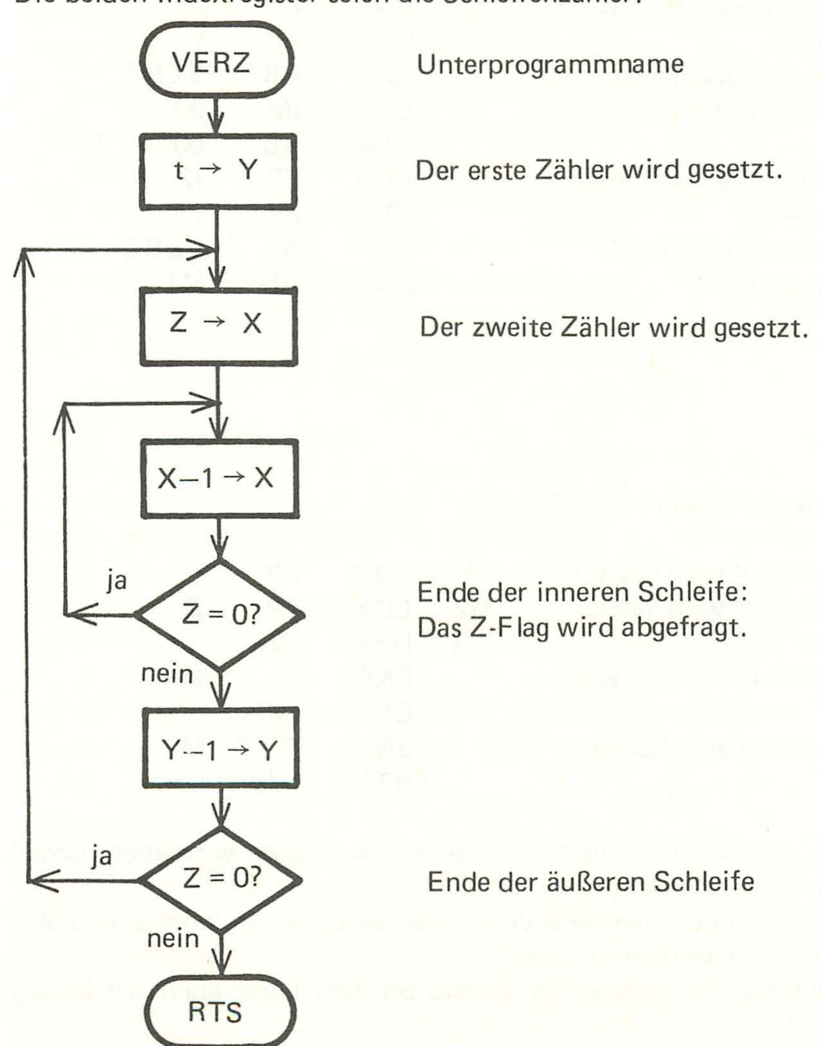
Es wird eine bestimmte Zahl t (Zeitkonstante) in das X-Register geladen. Die CPU durchläuft t mal die Schleife. Wenn X = 0 ist, verläßt sie die Programmschleife.

Will man eine Zeitverzögerung im Bereich von Millisekunden programmieren, muß man eine Doppelschleife anlegen.

Da wir zweimal eine Verzögerung programmieren müßten, formulieren wir ein entsprechendes Unterprogramm.

Das Verzögerungsunterprogramm:

Wir legen eine Zeitkonstante (t) variabel und eine Konstante (Z) fest. Die beiden Indexregister seien die Schleifenzähler.



Das Programm im Assemblercode:

Die Konstante t1, t2, t und Z werden im Datenspeicher (0. Seite) abgelegt.

Die Vorbereitung:		LDA	IM	FF	
		STA	AB	01	17
Bit 0 auf 1:	M1	LDA	IM	FF	
		STA	AB	00	17
Die Konstante t wird mit		LDA	ZP	t1	
t1 geladen:		STA	ZP	t	
Zeitverzögerung mit t1:		JSR	AB	VERZ	
Bit 0 wird gelöscht:		LDA	IM	00	
		STA	AB	00	17
Die Konstante t wird mit		LDA	ZP	t2	
t2 geladen:		STA	ZP	t	
Zeitverzögerung mit t2:		JSR	AB	VERZ	
Sprung nach M1:		JMP	AB	M1	

Das Unterprogramm VERZ:

Der Zähler Y wird gesetzt:		LDY	ZP	t
Der Zähler X wird gesetzt:	M2	LDX	ZP	Z
	M3	DEX	IMP	
Ende der inneren Schleife:		BNE	R	M3
		DEY	IMP	
Ende der äußeren Schleife:		BNE	R	M2
		RTS	IMP	

Wollen wir als Zeiteinheit 1ms wählen, so müssen wir experimentell die Konstante Z bestimmen.
Natürlich könnte man sie auch aus den einzelnen Befehlszeiten ableiten. Dies ist aber recht mühselig.
Die Übersetzung dieses Programmes sei dem Leser auch zur Übung empfohlen.

4.1.3 Grundprogramme zur Bedienung peripherer Ein- und Ausgabeeinheiten

Die Stärke eines frei programmierbaren Microcomputers liegt unter anderem darin, daß der Anwender nahezu beliebige periphere Geräte anschließen kann.

In den meisten Fällen ist ein Anschluß eines Gerätes ohne wesentlichen Hardwareaufwand möglich. Allerdings muß eine gerätespezifische Bedienungssoftware entwickelt werden. Einige Beispiele zum Anschluß einfacher peripherer Geräte werden in den folgenden Abschnitten vorgestellt. Der Hardwareaufwand ist, wie der Leser sicherlich bestätigen wird, extrem gering. Die Bedienungssoftware wird allerdings umfangreicher, wenn der Hardwareaufwand verkleinert wird.

Die Bedienungsprogramme kann man in zwei Gruppen einteilen.

1. Gruppe: Treiberprogramme
Das sind Programme, die die Hardware bedienen. Sie schalten zum Beispiel den Drucker ein, kontrollieren, ob er auch alle Zeichen druckt, wandeln parallele Daten in serielle um, achten auf das richtige Zeitverhalten (timing) u. s. w.
Die Treiberprogramme sind genau auf die verwendete Hardware zugeschnitten.

2. Gruppe: Codewandler
Da meist jedes periphere Gerät seinen speziellen Code besitzt, der nicht immer mit dem programminternen Code übereinstimmt, muß häufig eine Codewandlung durchgeführt werden, die ein spezielles Programm übernimmt.

Die Codewandler kann man als die eigentlichen Grundprogramme bezeichnen, da sie relativ geräteunabhängig sind.

In diesem Abschnitt wollen wir die wichtigsten Codewandler kennenlernen. Da sie universell sein sollen, werden sie als Unterprogramme konzipiert.

Dem Leser wird es überlassen, ein auf seine spezielle Hardware zugeschnittenes Testprogramm zu schreiben. Um ihm eine Hilfe zu geben, werden am Ende dieses Abschnittes einige Testprogramme vorgestellt.

Die Prinzipien der Codewandlung:

Es gibt berechnende Codewandlungsverfahren. Bei diesen Verfahren wird eine Rechenvorschrift (Algorithmus) benutzt, die die Umwandlung vom Quellcode in den Zielcode ermöglicht. Als Beispiel werden die Programme 4. und 5. vorgestellt.

Die meisten Codewandlungen verwenden ein Tabellenverfahren. Bei einem solchen Verfahren wird eine geeignete Tabelle (Feld) des Zielcodes angelegt und der Quellcode wird als Adresse (Index) der einzelnen Feldelemente interpretiert.

Nicht immer kann der Quellcode zur Adressierung der Zielcodetabelle herangezogen werden. Dieses Problem tritt bei einem unvollständigen Code auf, dessen Zeichenvorrat nicht gleichmächtig wie die möglichen Binärkombinationen seiner Dualdarstellung ist. Der ASCII-Code ist nicht vollständig, da er nicht 256 Zeichen besitzt. Seine Dualdarstellung (8 bit Darstellung) ermöglicht aber 256 Binärkombinationen.

Dann kann es notwendig sein, daß eine zweite Tabelle des Quellcodes angelegt wird und ein „Vergleichsverfahren" benutzt wird.

Es ist nicht möglich, weiter auf die speziellen Tabellenverfahren einzugehen, da der Umfang dieses Buches gesprengt würde.

1. Codewandlung Hex → Siebensegment (SEG)

Die Problemstellung:

Es soll eine Hexzahl auf einer Siebensegmentanzeige dargestellt werden.

Die Problemlösung:

Die einzelnen Segmente werden nach Tabelle 4.1.3.1 angesteuert. Man kann die einzelnen Schaltzustände natürlich als eine 8 bit Dualzahl verstehen.

Die Hexcodierung dieser Tabelle wird in den Speicher als Feld eingetragen. Wie man sofort erkennen kann, ist der Quellcode als Zugriffsadresse zu diesem Feld erklärbar. Zum Beispiel entspricht die Hexzahl A dem 10. Element dieses Feldes und die Hexzahl 0 hat als Zielcode das 0. Element des Feldes.

Der Zugriff zu diesem Feld kann durch ein extrem kleines Unterprogramm ausgeführt werden.

1. Eingabeparameter:	Zahl im Hexcode im X-Register
2. Ausgabeparameter:	Zahl im 7-Segmentcode in Register A

Der Zugriff geschieht durch einen absolut durch X indizierten LDA-Befehl.

Das Programm im Maschinencode (Hexcode):

Das Datenfeld ist ab Adresse 0204 untergebracht.

```
0200  bd  lda abx 04  02
0203  60  rts imp
```

Das Datenfeld:

```
0204 3f 06 5b 4f 66 6d 7d 07 7f 67 77 7c 39 5e 79 71
```

Tabelle 4.1.3.1

Hexzahl	Segment G	F	E	D	C	B	A	Hex
0	0	1	1	1	1	1	1	3F
1	0	0	0	0	1	1	0	06
2	1	0	1	1	0	1	1	5B
3	1	0	0	1	1	1	1	4F
4	1	1	0	0	1	1	0	66
5	1	1	0	1	1	0	1	6D
6	1	1	1	1	1	0	1	7D
7	0	0	0	0	1	1	1	07
8	1	1	1	1	1	1	1	7F
9	1	1	0	0	1	1	1	67
A	1	1	1	0	1	1	1	77
B	1	1	1	1	1	0	0	7C
C	0	1	1	1	0	0	1	39
D	1	0	1	1	1	1	0	5E
E	1	1	1	1	0	0	1	79
F	1	1	1	0	0	0	1	71

2. Codewandlung Hex → 5 Kanal Fernschreibcode (HFS)

Der 5 Kanal Fernschreibcode ist ein 5 bit Code. Der volle Zeichensatz wird in zwei Gruppen geteilt (Ziffern und Buchstaben). Das Treiberprogramm muß Ziffern und Buchstaben unterscheiden können und ein Umschaltzeichen senden, wenn ein Zeichenartwechsel auftritt. Daher wird maschinenintern der 5 bit Code zu einem 6 bit Code erweitert. Das 6. Bit ist 1, wenn eine Ziffer vorliegt, sonst 0.

In Tabelle 4.1.3.2 ist die Wandlungstabelle abgebildet. Diese wird als Feld in den Speicher geladen.

Das Wandlungsprogramm entspricht genau dem des 1. Beispiels.

Tabelle 4.1.3.2

Hexziffer	Bit						Hex
	5	4	3	2	1	0	
0	1	1	0	1	1	0	36
1	1	1	0	1	1	1	37
2	1	1	0	0	1	1	33
3	1	0	0	0	0	1	21
4	1	0	1	0	1	0	2A
5	1	1	0	0	0	0	30
6	1	1	0	1	0	1	35
7	1	0	0	1	1	1	27
8	1	0	0	1	1	0	26
9	1	1	1	0	0	0	38
A	0	0	0	0	1	1	03
B	0	1	1	0	0	1	19
C	0	0	1	1	1	0	0E
D	0	0	1	0	0	1	09
E	0	0	0	0	0	1	01
F	0	0	1	1	0	1	0D

Das Programm:

```
0200  bd  lda abx 04  02
0203  60  rts imp
```

Das Feld:

```
0204 36  37  33  21  2a  30  35  27  26  38  03  19  0e  09  01  0d
```

3. Codewandlung 5 Kanal FS-Code → Hex (FSH)

Soll ein 5 Kanal Fernschreiber als Eingabestation benutzt werden, ist auch eine umgekehrte Codewandlung notwendig.

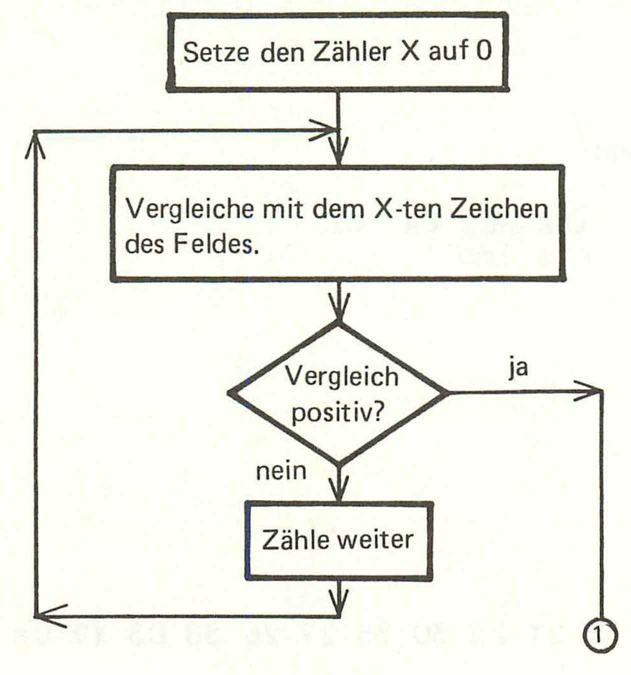
Die Problemanalyse:

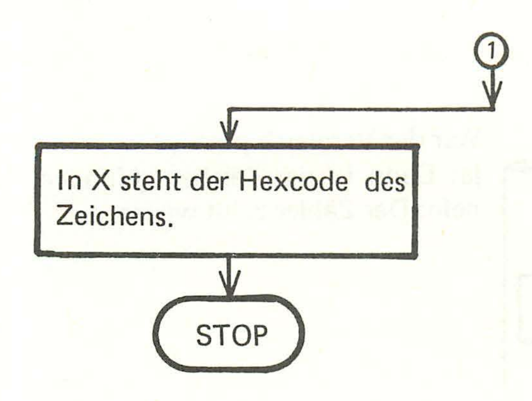
Aus Tabelle 4.1.3.2 ist ersichtlich, daß der 5 Kanal Code nicht die gleiche Anordnung besitzt, wie die zugehörigen Ziffern und Buchstaben.
Zum Beispiel: 3 ist größer als 2.
Aber 00001 ist kleiner als 10011.

Daher ist es nicht möglich, die Codewandlung wie in den vorherigen Beispielen durchzuführen.

Wenn wir allerdings den FS-Code in einem Feld in der Anordnung der Hexziffern anlegen (wie in Tabelle 4.1.3.2), so kann man ein Zählverfahren anwenden.

Die wesentlichen Schritte eines solchen Zählverfahrens sähen so aus:



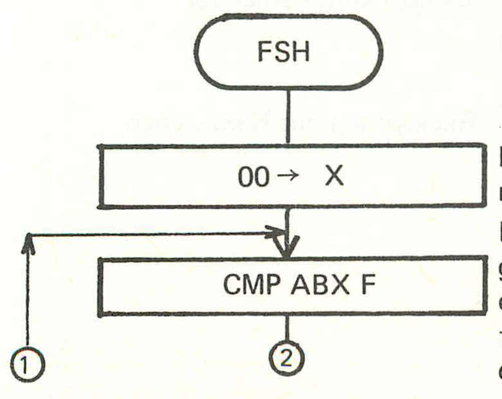


In unserem Codewandlungsprogramm sollten wir allerdings noch eine Fehlerdiagnose vorsehen, da es sehr viele nicht als Hexzahlen interpretierbare Zeichen gibt.

Die Problemlösung:

Wir legen das Programm als Unterprogramm an. Wenn ein fehlerhaftes Zeichen auftritt, soll C = 1 gesetzt werden, im anderen Fall soll C rückgesetzt werden.

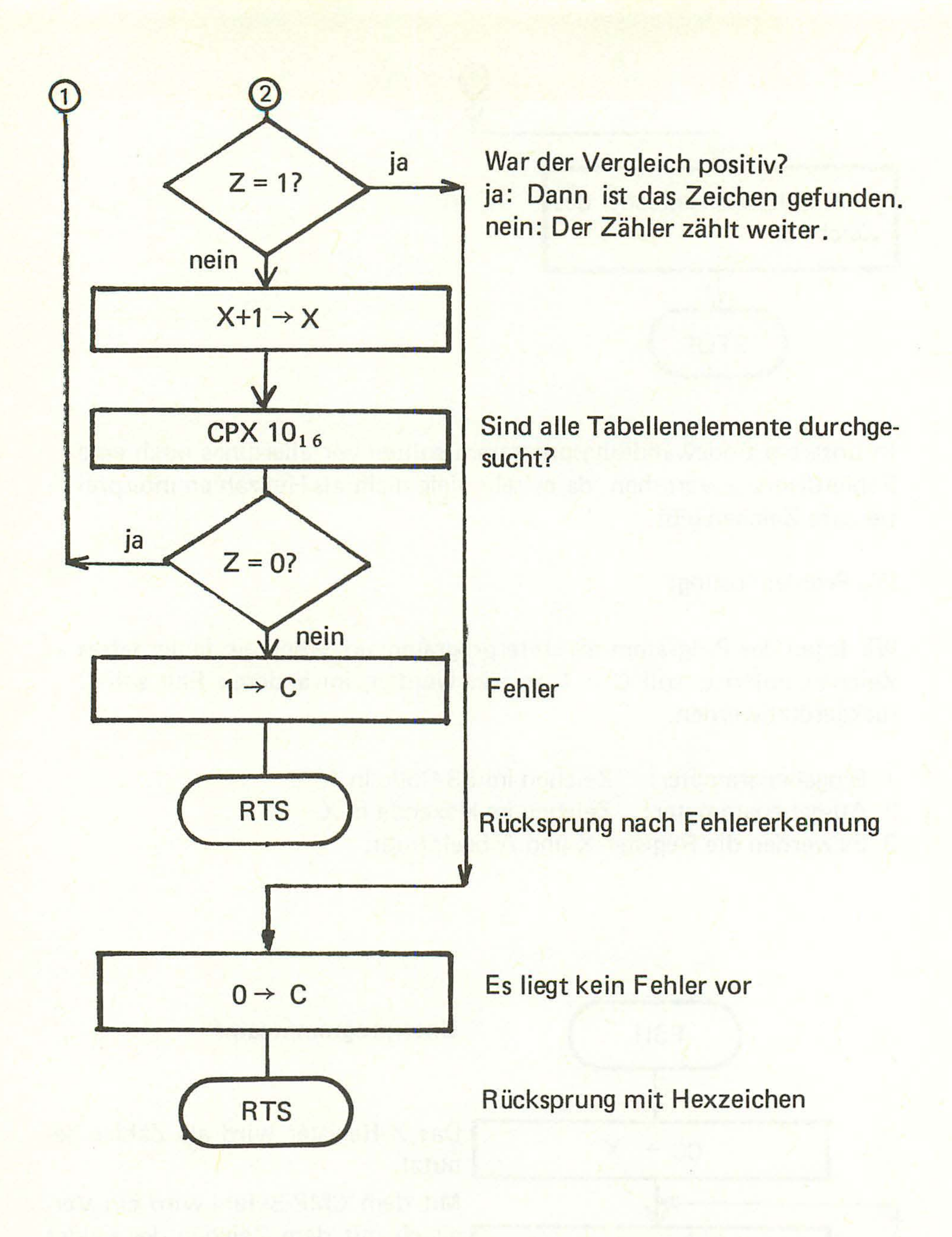
1. Eingabeparameter: Zeichen im FS-Code in A
2. Ausgabeparameter: Zeichen im Hexcode in X
3. Es werden die Register X und A beeinflußt.



Unterprogrammname

Das X-Register wird als Zähler benutzt.

Mit dem CMP-Befehl wird ein Vergleich mit dem Zeichen des Feldes durchgeführt, das den Index X trägt. Bei diesem Vergleich bleibt der Inhalt von A erhalten.



1
2
Z = 1?
ja
nein
X+1 → X
CPX 10₁₆
ja
Z = 0?
nein
1 → C
RTS
0 → C
RTS
War der Vergleich positiv?
ja: Dann ist das Zeichen gefunden.
nein: Der Zähler zählt weiter.
Sind alle Tabellenelemente durchgesucht?
Fehler
Rücksprung nach Fehlererkennung
Es liegt kein Fehler vor
Rücksprung mit Hexzeichen

Das Programm im Maschinencode (Hexcode):

Das Feld vom 2. Beispiel findet Verwendung.

```
0214  a2  ldx im   00
0216  dd  cmp abx  04  02
0219  f0  beq r    07
021b  e8  inx imp
021c  e0  cpx im   10
021e  d0  bne r    f6
0220  38  sec imp
0221  60  rts imp
0222  18  clc imp
0223  60  rts imp
```

Der Speicherauszug

```
0214 a2 00 dd 04 02 f0 07 e8 e0 10 d0 f6 38 60 18 60
```

Tabelle 4.1.3.3

Hexziffer	ASCII dual	hex
0	10110000	B0
1	10110001	B1
2	10110010	B2
3	10110011	B3
4	10110100	B4
5	10110101	B5
6	10110110	B6
7	10110111	B7
8	10111000	B8
9	10111001	B9
A	11000001	C1
B	11000010	C2
C	11000011	C3
D	11000100	C4
E	11000101	C5
F	11000110	C6

4. Codewandlung Hex → ASCII-Code (HASCII)

Die Codewandlung vom Hexcode in den amerikanischen 8-Kanal-Fernschreibcode und zurück ist leichter und schneller durchführbar, da man ähnliche Anordnungen entdecken kann. Die Tabelle 4.1.3.3 zeigt den uns interessierenden Ausschnitt aus der ASCII-Codetabelle.

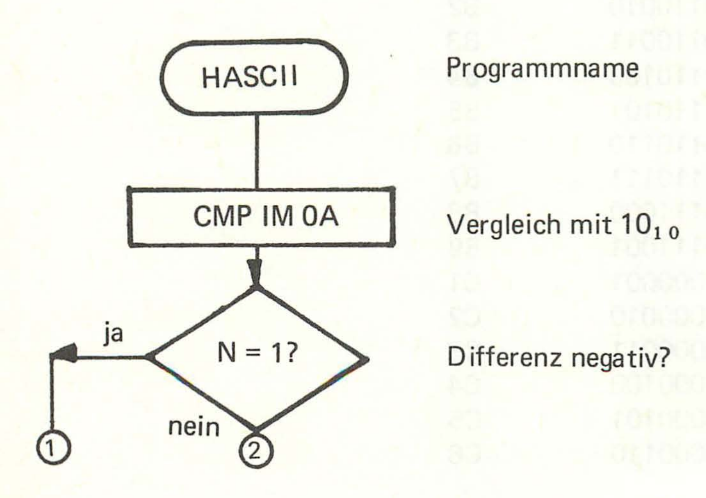
Der Code wird als 8 bit Code verstanden. Der 7 bit Code ignoriert einfach das 8. Bit.

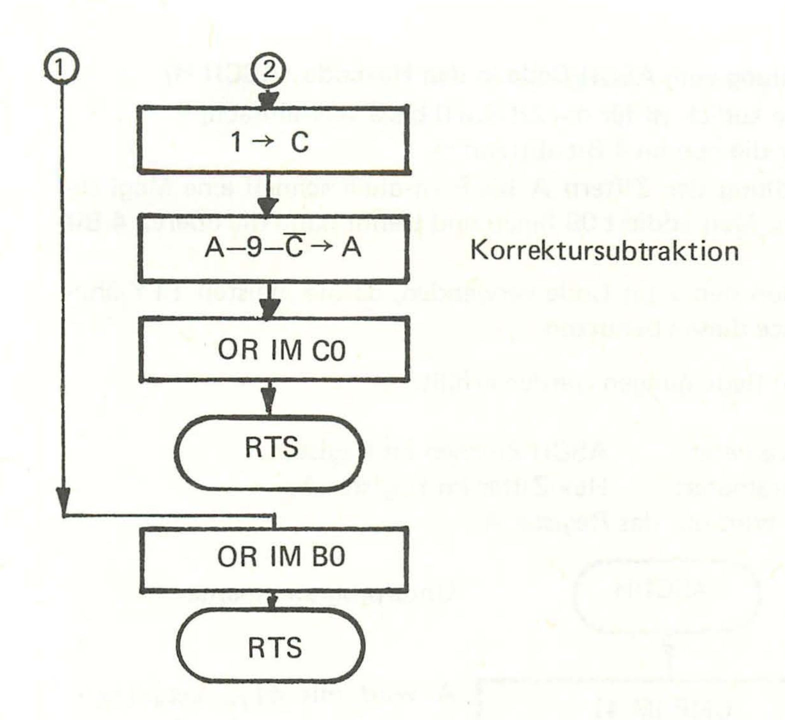
Die Problemlösung:

Wir unterteilen die Codewandlung in zwei Teilaufgaben.

1. Fall: Die Zahl liegt zwischen 0 und 9.
   Dann muß zum Dualcode die Hexzahl B0 hinzugefügt werden (OR-Verknüpfung).
2. Fall: Die Zahl ist größer als 9. Dann muß 9 subtrahiert und die Hexzahl C0 hinzugefügt werden.

Das Unterprogramm erfüllt folgende Bedingungen.
1. Eingabeparameter: Hexzahl im Register A.
2. Ausgabeparameter: ASCII-Code der Zahl im Register A.
3. Es beeinflußt das Register A.





Das Programm im Maschinencode (Hexcode):

```
02  00  C9  CMP  IM  0A
    02  30  BMI  R   06
    04  38  SEC  IMP
    05  E9  SBC  IM  09
    07  09  ORA  IM  C0
    09  60  RTS  IMP
    0A  09  ORA  IM  B0
    0C  60  RTS  IMP
```

Der Speicherauszug:

0200 C9 0A 30 06 38 E9 09 09 C0 60 09 B0 60

5. Codewandlung vom ASCII-Code in den Hexcode (ASCII H)

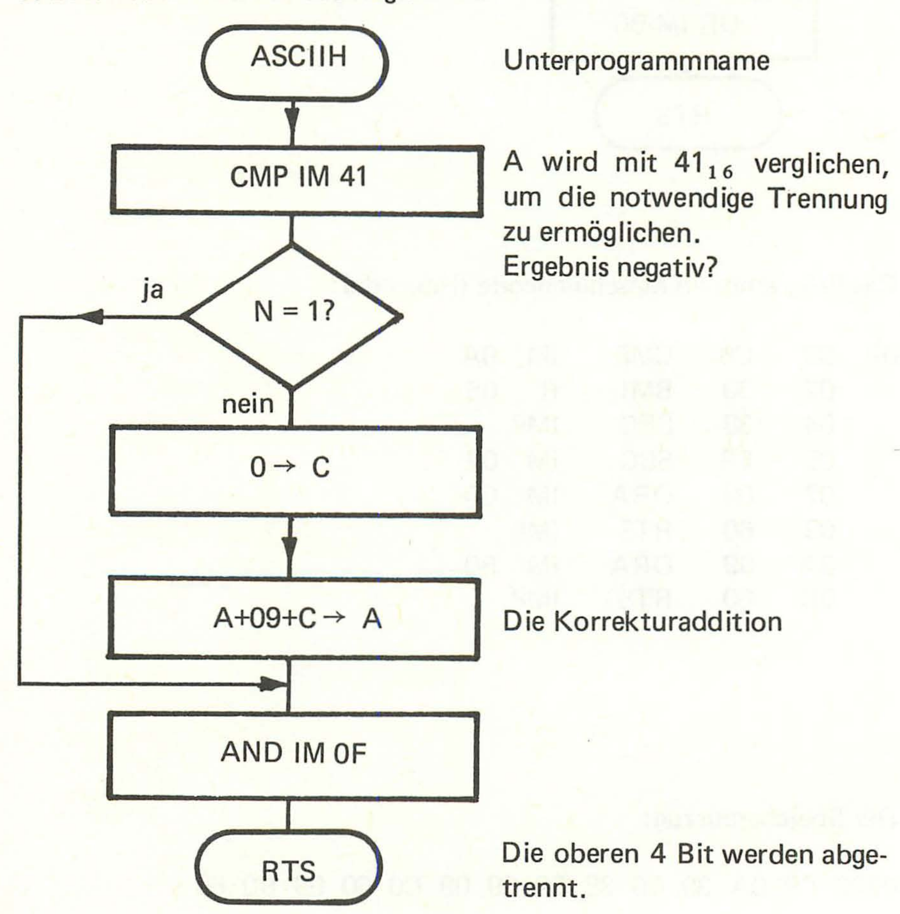
Die Wandlung zurück ist für die Ziffern 0 bis 9 sehr einfach.
Man muß nur die oberen 4 Bit abtrennen.

Für die Wandlung der Ziffern A bis F ist auch schnell eine Möglichkeit gefunden. Man addiert 09 hinzu und trennt dann die oberen 4 Bit ab.

Wir wollen nun den 7 bit Code verwenden, da die meisten TTY-ähnlichen Interface diesen benutzen.

Die folgenden Bedingungen werden erfüllt:

1. Eingabeparameter: ASCII-Zeichen im Register A
2. Ausgabeparameter: Hex-Ziffer im Register A.
3. Beeinflußt wird nur das Register A.



Das Programm im Maschinencode (Hexcode)

```
0200  c9  cmp im  41
0202  30  bmi r   03
0204  18  clc imp
0205  69  adc im  09
0207  29  and im  0f
0209  60  rts imp
```

Der Speicherauszug:

```
0200  c9 41 30 03 18 69 09 29 0f 60
```

6. Codewandlung vom 5-Kanal-FS-Code in den ASCII-Code (5K ASCII)

Der Standardzeichencode der meisten Microcomputersoftware ist der amerikanische ASCII-Code. Soll ein europäischer 5-Kanal-Fernschreiber als Datenstation verwendet werden, muß zwischen der Software und dem Treiber ein geeigneter Codewandler eingefügt werden.

Da ältere Fernschreiber recht preiswert erhältlich sind, ist für viele Leser sicherlich die Vorstellung solcher Codewandler von Interesse.

Es tritt bei der Codewandlung ein grundsätzliches Problem auf: Der Zeichenvorrat des ASCII-Code ist wesentlich umfangreicher als der des 5-Kanal-Codes. Bei der Wandlung vom ASCII-Code in den 5-Kanal-Code muß man darauf achten, daß Zeichen, die kein Bild im 5-Kanal-Code besitzen, irgendein beliebiges Zeichen zugeordnet wird.

Die Codetabellen beider Codes sind in Kapitel 2.1 zu finden.

In Tabelle 4.1.3.4 wurde die Zuordnung von 5-Kanal in ASCII ausgeführt. Die Tabelle ist nach der Anordnung der Zeichen im 5-Kanalcode angeordnet. Wir fügen ein 6. Bit hinzu, um die Ziffern von den Buchstaben unterscheiden zu können.

Wenn wir uns die Tabelle näher anschauen, entdecken wir, daß der 5-Kanalcode nicht ganz vollständig ist. Für einige Dualzahlen existiert kein Zeichen. Wir zählen neun Lücken. Dies ist noch eine geringe Anzahl.

So erscheint trotzdem das Tabellen-Adress-Verfahren als günstigstes Codewandlungsverfahren.

Wir legen die Tabelle in ein Feld F ab, dessen Adressen der 5-Kanalcode der entsprechenden Zeichen sind.

Wir können nun das Programm formulieren.

1. Eingabeparameter: Zeichen im 5-Kanalcode im Register A.
2. Ausgabeparameter: Zeichen im ASCII-Code im Register A.
3. Beeinflußt werden die Register X und A.

Da wir die indizierte Adressierung verwenden, muß das Zeichen nach X gebracht werden.

Das Programm im Maschinencode (Hexcode):

```
0200   aa   tax imp
0201   bd   lda abx 04  02
0204   60   rts imp
```

Das Feld F:

```
0205 c5 8a c1 a0 d3 c9 d5 8d c4 d2 ca ce c6 c3 cb d4
0215 da cc d7 c8 d9 d0 d1 cf c2 c7 00 cd d8 d6 00 00
0225 b3 8a ad a0 a7 b8 b7 8d 00 b4 00 ac 00 ba a8 b5
0235 ab a9 b2 00 b6 b0 b1 b9 bf 00 00
```

Tabelle 4.1.3.4: Umwandlung 5-Kanal nach ASCII

Dualzahl	Zeichen	ASCII (hex)	Dualzahl	Zeichen	ASCII (hex)
111110	=	BD	011110	V	D6
111101	/	AF	011101	X	D8
111100	.	AE	011100	M	CD
111001	?	BF	011010	G	C7
111000	9	B9	011001	B	C2
110111	1	B1	011000	0	CF
110110	0	B0	010111	Q	D1
110101	6	B6	010110	P	D0
110011	2	B2	010101	Y	D9
110010	)	A9	010100	H	C8
110001	+	AB	010011	W	D7
110000	5	B5	010010	L	CC
101111	(	A8	010001	Z	DA
101110	:	BA	010000	T	D4
101100	,	AC	001111	K	CB
101011	Glocke		001110	C	C3
101010	4	B4	001101	F	C6
101001	Wer da?		001100	N	CE
101000	CR	8D	001011	J	CA
100111	7	B7	001010	R	D2
100110	8	B8	001001	D	C4
100101	'	A7	001000	CR	8D
100100	ZR	A0	000111	U	D5
100011	–	AD	000110	I	C9
100010	LF	8A	000101	S	D3
100001	3	B3	000100	ZR	A0
			000011	A	C1
			000010	LF	8A
			000001	E	C5

7. Codewandlung von ASCII in 5-Kanal (ASCII 5K)

Tabelle 4.1.3.5 gibt die hierfür entsprechende Zuordnungsvorschrift an.

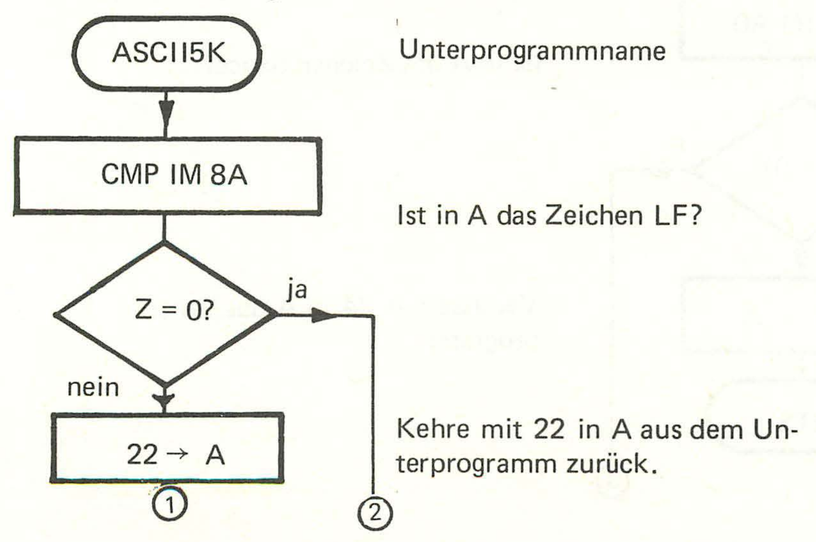
Ab dem ASCII-Zeichen + (AB) sind die Dualdarstellungen lückenlos. AA wird einfach die „Glocke" zugeordnet. So ist der Bereich ab A7 lückenlos.

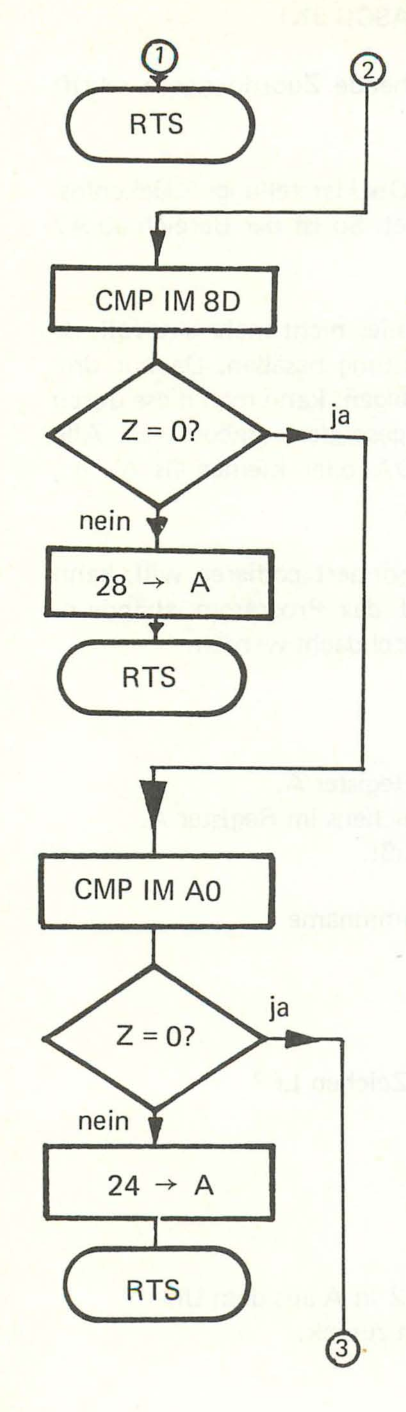
Ein reines Tabellen-Adress-Verfahren ist hier nicht mehr sinnvoll, da immerhin 28 Tabellenstellen keine Bedeutung besäßen. Da nur drei Zeichen sich nicht in die Reihenfolge einfügen, kann man diese durch eine besondere Abfrage entdecken und gesondert umcodieren. Alle ASCII-Zeichen, deren Code größer als DA oder kleiner als A7 ist, sollen durch die „Glocke" codiert werden.

Derjenige Leser, der weitere Zeichen gesondert codieren will, kann die Tabelle entsprechend erweitern und das Programm abändern. Das Programmierprinzip muß nicht neu durchdacht werden.

Die Bedingungen des Unterprogramms:

1. Eingabeparameter: ASCII-Zeichen im Register A.
2. Ausgabeparameter: 5-Kanalcode des Zeichens im Register A.
3. Es werden die Register A und X beeinflußt.



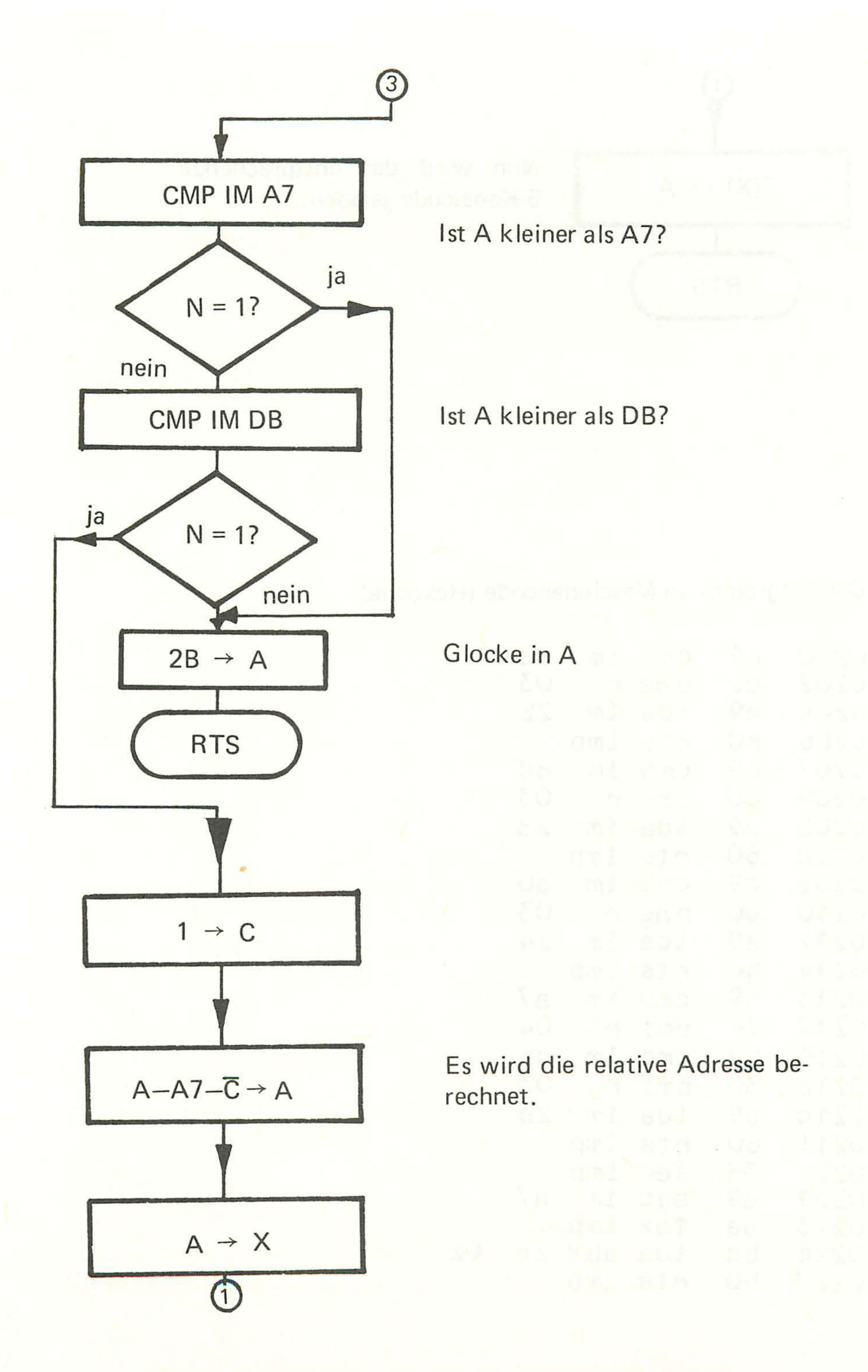


Ist in A das Zeichen CR?

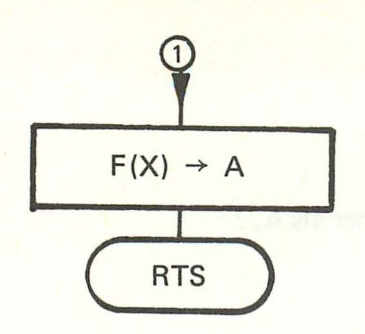
Verlasse mit 28 in A das Unterprogramm.

Ist in A das Zeichen Space?

Verlasse mit 24 in A das Unterprogramm.



3
CMP IM A7
Ist A kleiner als A7?
ja
N = 1?
nein
CMP IM DB
Ist A kleiner als DB?
ja
N = 1?
nein
2B → A
Glocke in A
RTS
1 → C
A–A7–C̄ → A
Es wird die relative Adresse berechnet.
A → X
1



Nun wird der entsprechende 5-Kanalcode geladen.

Das Programm im Maschinencode (Hexcode):

```
0200   c9   cmp im  8a
0202   d0   bne r   03
0204   a9   lda im  22
0206   60   rts imp
0207   c9   cmp im  8d
0209   d0   bne r   03
020b   a9   lda im  28
020d   60   rts imp
020e   c9   cmp im  a0
0210   d0   bne r   03
0212   a9   lda im  24
0214   60   rts imp
0215   c9   cmp im  a7
0217   30   bmi r   04
0219   c9   cmp im  db
021b   30   bmi r   03
021d   a9   lda im  2b
021f   60   rts imp
0220   38   sec imp
0221   e9   sbc im  a7
0223   aa   tax imp
0224   bd   lda abx 28  02
0227   60   rts imp
```

Der Speicherauszug:

```
0200 c9 8a d0 03 a9 22 60 c9 8d d0 03 a9 28 60 c9 a0
0210 d0 03 a9 24 60 c9 a7 30 04 c9 db 30 03 a9 2b 60
0220 38 e9 a7 aa bd 28 02 60
```

Das Feld:

```
0228 25 2f 32 2b 31 2c 23 3c 3d 36 37 33 21 2a 30 35
0238 27 26 38 2e 2b 2b 3e 2b 39 2b 03 19 0e 09 01 0d
0248 1a 14 06 0b 0f 12 1c 0c 18 16 17 0a 05 10 07 1e
0258 13 1d 15 11
```

Tabelle 4.1.3.5: Umwandlung ASCII in 5-Kanal

ASCII	Zeichen	5-Kanal (hex)	ASCII	Zeichen	5-Kanal (hex)
8A	LF	22	C1	A	03
8D	CR	28	C2	B	19
A0	Space	24	C3	C	0E
A7	'	25	C4	D	09
A8	(	2F	C5	E	01
A9	)	32	C6	F	0D
AA	Glocke	2B	C7	G	1A
AB	+	31	C8	H	14
AC	,	2C	C9	I	06
AD	–	23	CA	J	0B
AE	.	3C	CB	K	0F
AF	/	3D	CC	L	12
B0	0	36	CD	M	1C
B1	1	37	CE	N	0C
B2	2	33	CF	0	18
B3	3	21	D0	P	16
B4	4	2A	D1	Q	17
B5	5	30	D2	R	0A
B6	6	35	D3	S	05
B7	7	27	D4	T	10
B8	8	26	D5	U	07
B9	9	38	D6	V	1E
BA	:	2E	D7	W	13
BD	=	3E	D8	X	1D
BF	?	39	D9	Y	15
			DA	Z	11

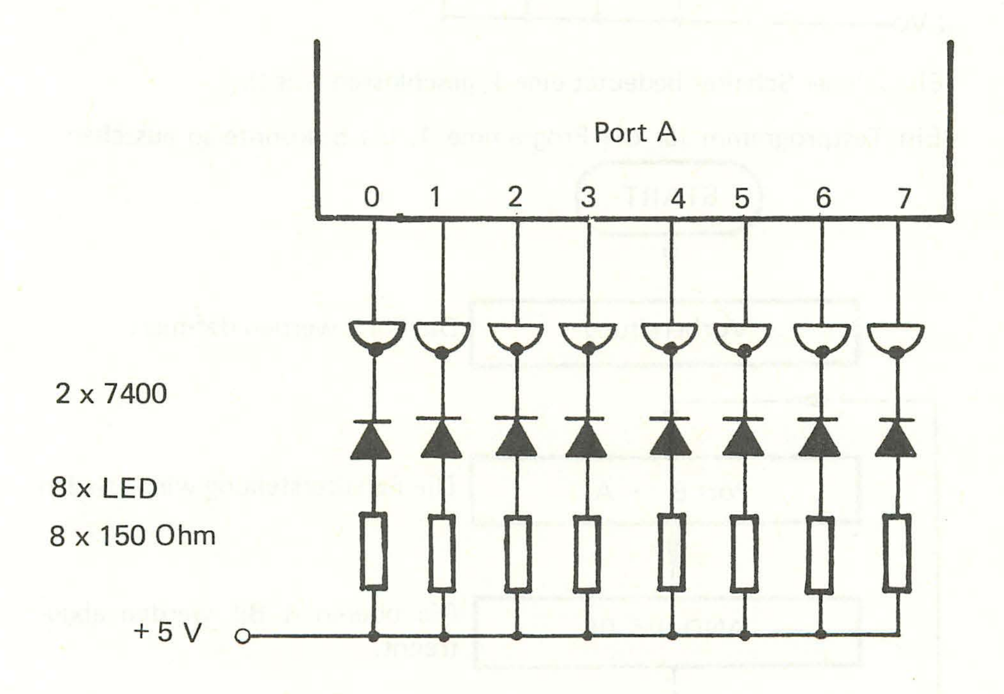
8. Die Testprogramme

Natürlich wollen wir die besprochenen Codewandler testen. Hierzu müssen wir noch ein kleines Testprogramm schreiben.

Die Struktur dieses Programms hängt wesentlich von der Hardware ab, die uns zur Verfügung steht.

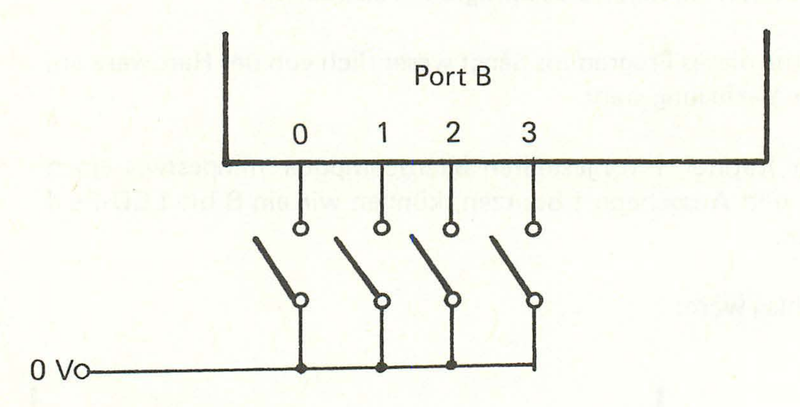
Da alle in Kapitel 1 vorgestellten Microcomputer mindestens einen 8 bit Ein- und Ausgabeport besitzen, können wir ein 8 bit LED-Feld anschließen.

Ein Vorschlag wäre:



Der KIM besitzt noch einen weiteren Port, der für den Anwender leicht zugänglich ist, das ist Port B mit den Adressen PBD 1702 und PBDD 1703. PBD ist das Datenregister und PBDD das Datenrichtungsregister.

Dort schließen wir ein Schalterfeld mit vier Schaltern an.



Ein offener Schalter bedeutet eine 1, geschlossen eine 0.

Ein Testprogramm für die Programme 1. bis 5. könnte so aussehen:

START

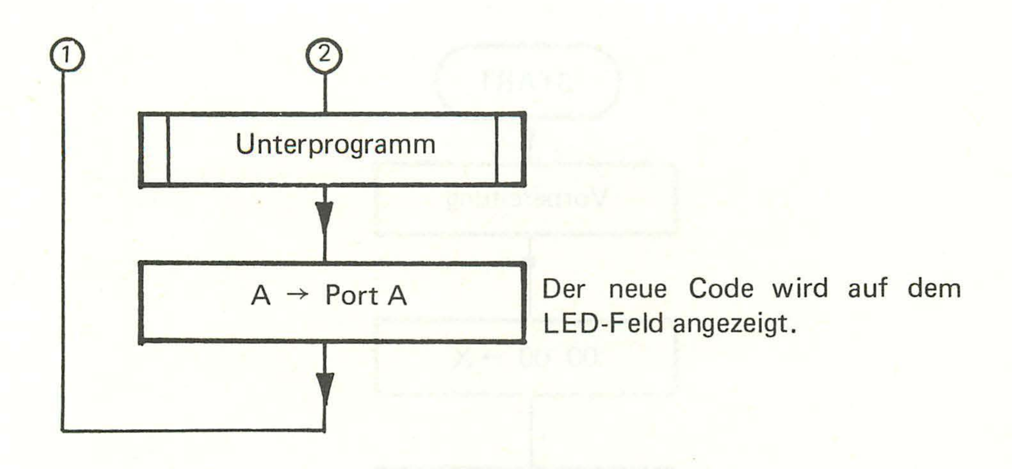
Vorbereitung — Die Ports werden definiert

Port B → A — Die Schalterstellung wird geladen

AND IM 0F — Die oberen 4 Bit werden abgetrennt.

A → X — Die Daten müssen zur Übergabe ins X-Register gegeben werden.

① ②

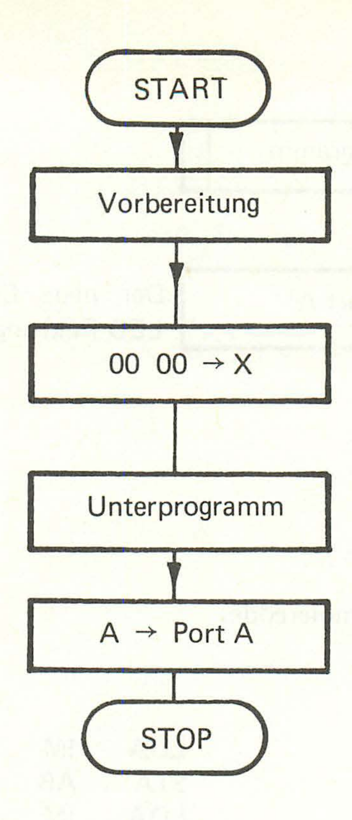


Das Programm im Assemblercode:

Die Vorbereitung:		LDA	IM	FF
		STA	AB	PADD
		LDA	IM	00
		STA	AB	PBDD
Die Datenübernahme:	M	LDA	AB	PBD
		AND	IM	0F
		TAX	IMP	
Sprung in das Unterprogramm:		JSR	AB	00 02
Übergabe an das LED–Feld:		STA	AB	PAD
		JMP	AB	M

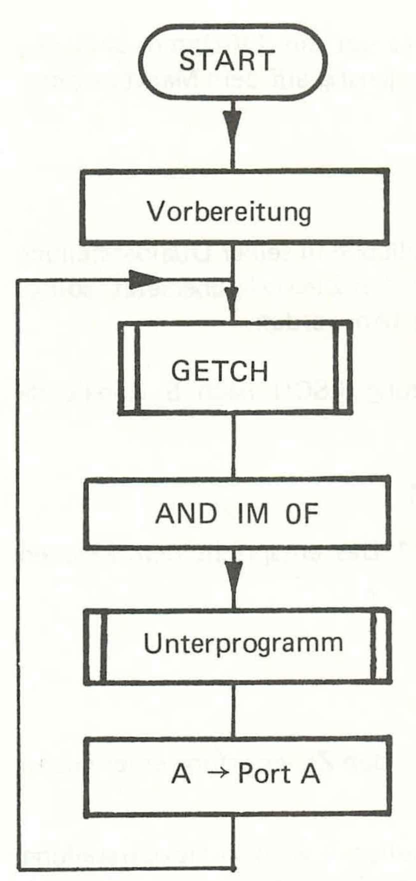
Man könnte die Datenübernahme auch anders gestalten.

Zum Beispiel:
In die Speicherzelle 00 00 wird die entsprechende Hexzahl abgelegt und anschließend das Programm gestartet, das diese Hexzahl in das X-Register übernimmt.



Falls uns ein TTY oder eine ähnliche ASCII-Datenstation zur Verfügung steht, können wir diese auch zum Testen einsetzen. Am Beispiel des KIM soll es erläutert werden.

Wir schließen an den Port A das LED-Feld an und benutzen die ASCII-Station als Dateneingabestation. Da das Betriebsprogramm des KIM in Unterprogrammen die Datenannahme ermöglicht, ist wenig Programmierarbeit zu leisten.



Ein Unterprogramm des KIM-Monitors, das die Übernahme eines Zeichens besorgt. Es hat die Startadresse 1E5A.

Beim Testen des 6. Programmes muß diese Anweisung entfallen.

Das Programm im Assemblercode:

```
     LDA   IM   FF
     STA   AB   PADD
M    JSR   AB   5A 1E
     AND   IM   0F
     JSR   AB   00 02
     STA   AB   PAD
     JMP   AB   M
```

Es soll nun ein umfangreicheres Testprogramm vorgestellt werden, das mit einer ASCII-Datenstation arbeitet und keine weitere Hardware erfordert. Es ist für das Testen umfangreicher Codewandler besonders gut geeignet.

Als Datenstation eignet sich besonders gut ein TV-Datensichtgerät. Solche Geräte werden schon recht preisgünstig auf dem Markt angeboten.

Die Problemstellung:

Es soll ein bestimmtes Zeichen im Quellcode in seiner Dualdarstellung ausgedruckt werden. Anschließend, in den Zielcode übersetzt, soll es wieder in seiner Dualdarstellung ausgegeben werden.

Zum Beispiel wird bei der Übersetzung ASCII nach 5 Kanalcode ausgedruckt:

Im Quellcode:	C1	1100 0001	
Im Zielcode	03	0000 0011	Das entspricht dem Zeichen A

Die Problemanalyse:

1. Schritt:	Der Drucker wird auf den Zeilenanfang einer neuen Zeile gesetzt.
2. Schritt:	Das Zeichen im Quellcode wird in Hexdarstellung eingegeben und gleichzeitig gedruckt.
3. Schritt:	Es werden zwei Leerzeichen gedruckt.
4. Schritt:	Das betreffende Zeichen wird in seiner Dualdarstellung angegeben.
5. Schritt:	Der Drucker wird auf den Zeilenanfang einer neuen Zeile gesetzt.
6. Schritt:	Das Zeichen wird in den Zielcode übersetzt.
7. Schritt:	Der Zielcode wird in Hexdarstellung ausgedruckt.
8. Schritt:	Es folgen zwei Leerzeichen.
9. Schritt:	Das Zeichen im Zielcode wird in dualer Form ausgegeben.

Es folgt nun wieder der 1. Schritt und es kann ein neues Zeichen eingetippt werden.

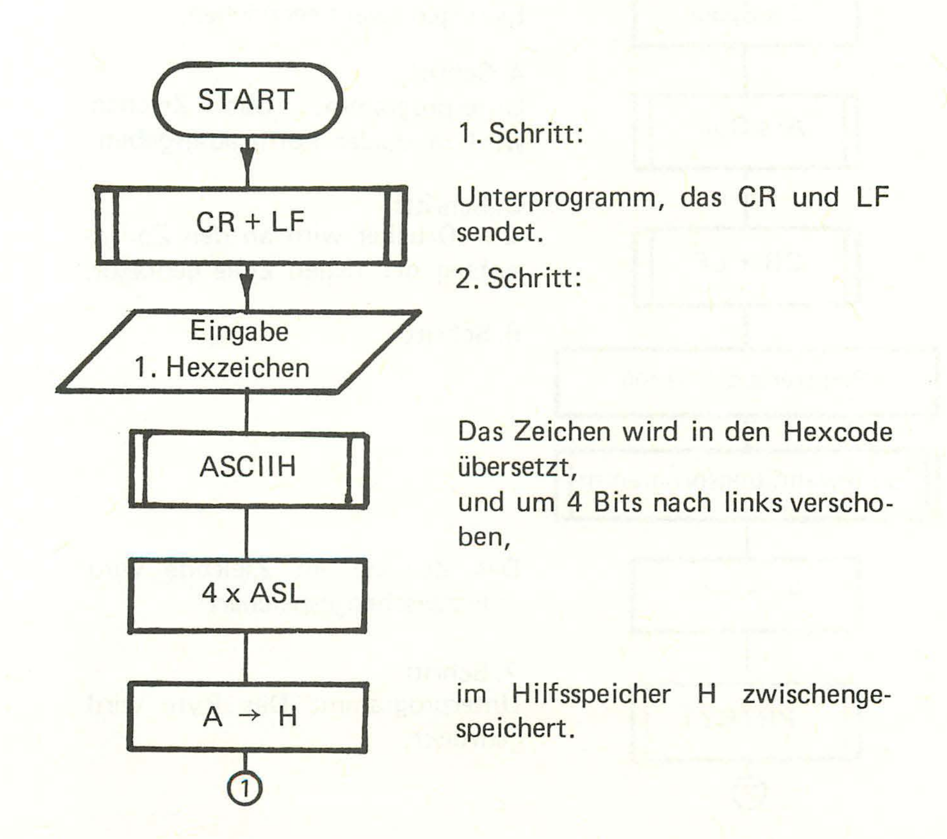
Die Problemlösung:

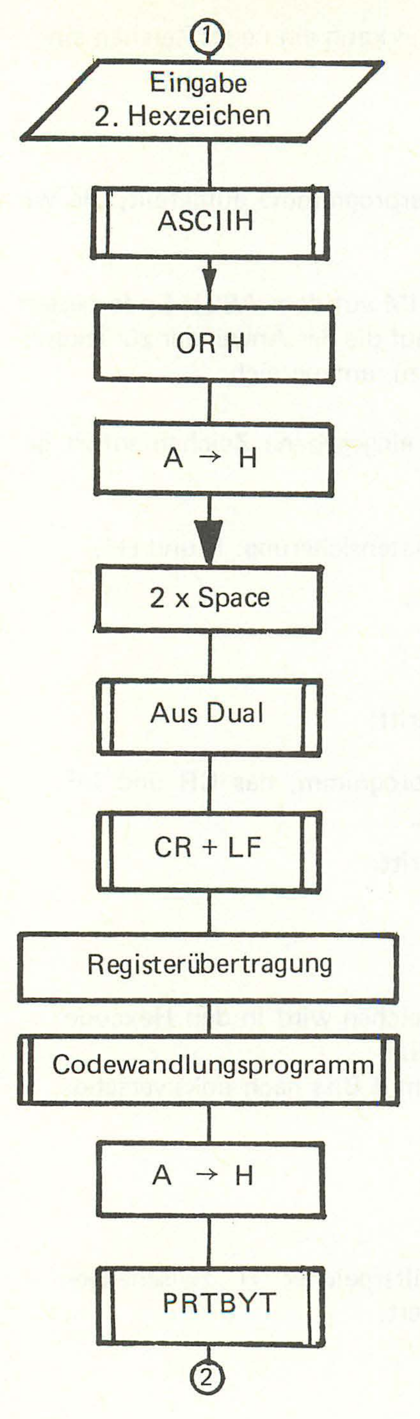
Das Programm wird in mehrere Unterprogramme aufgeteilt, die wir anschließend näher besprechen.

Da das TTY-Betriebsprogramm des KIM auf dem ASCII-Code basiert und einige Unterprogramme besitzt, auf die der Anwender zurückgreifen kann, wird das Programm nicht allzu umfangreich.

Das Interface ist so aufgebaut, daß eingegebene Zeichen sofort gedruckt werden.

Wir benötigen zwei Hilfsspeicher zur Datensicherung: H und H1.





Das zweite Hexzeichen wird übersetzt,

mit dem 1. Zeichen verknüpft,

und als vollständiges Zeichen im Quellcode in H abgespeichert.

3. Schritt:

Es folgen zwei Leerzeichen.

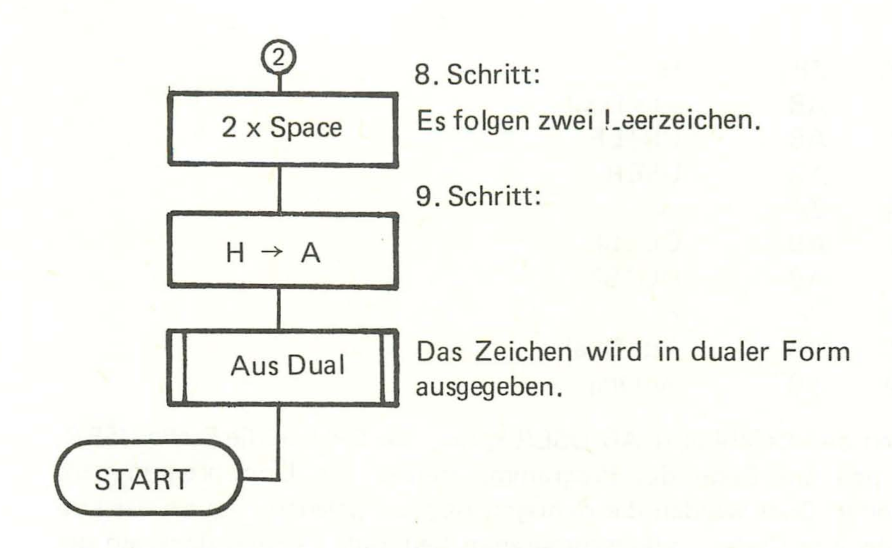
4. Schritt:
Unterprogramm: Das Zeichen wird in dualer Form ausgegeben.

5. Schritt:
Der Drucker wird an den Zeilenanfang der neuen Zeile gebracht.

6. Schritt:

Das Zeichen im Zielcode wird in H zwischengespeichert.

7. Schritt:
Unterprogramm: Das Byte wird gedruckt.



8. Schritt:

Es folgen zwei Leerzeichen.

9. Schritt:

Das Zeichen wird in dualer Form ausgegeben.

Das Aussenden zweier Leerzeichen (2 x Space) läßt sich einfach durch zwei Unterprogrammsprünge in das KIM-Monitorunterprogramm OUTSP verwirklichen. Daher wollen wir dafür kein spezielles Unterprogramm schreiben.

Die Eingabe eines ASCII-Zeichens erfolgt ebenfalls durch ein Unterprogramm im Monitor mit dem Namen GETCH.

Das Programm im Assemblercode:

```
Anfang  JSR   AB    CR+LF
        JSR   AB    GETCH
        JSR   AB    ASCIIH
        ASL   AC
        ASL   AC
        ASL   AC
        ASL   AC
        STA   ZP    H
        JSR   AB    GETCH
        JSR   AB    ASCIIH
        ORA   ZP    H
        STA   ZP    H
        JSR   AB    OUTSP
        JSR   AB    OUTSP
```

```
LDA   ZP   H
JSR   AB   Aus Dual
JSR   AB   CR+LF
JSR   AB   USER
STA   ZP   H
JSR   AB   OUTSP
JSR   AB   OUTSP
LDA   ZP   H
JSR   AB   Aus Dual
JMP   AB   Anfang
```

Durch den Befehl JSR AB USER springt die CPU an die Stelle USER, die sich am Ende des Programms (hinter den Unterprogrammen) befindet. Dort werden die richtigen Register geladen, wie sie das entsprechende Codewandlungsprogramm benötigt. Es folgt dann ein unbedingter Sprung in dieses Codewandlungsprogramm. Der Rücksprungbefehl führt die CPU dann wieder an die richtige Stelle im Hauptprogramm zurück.

USER könnte so aussehen:

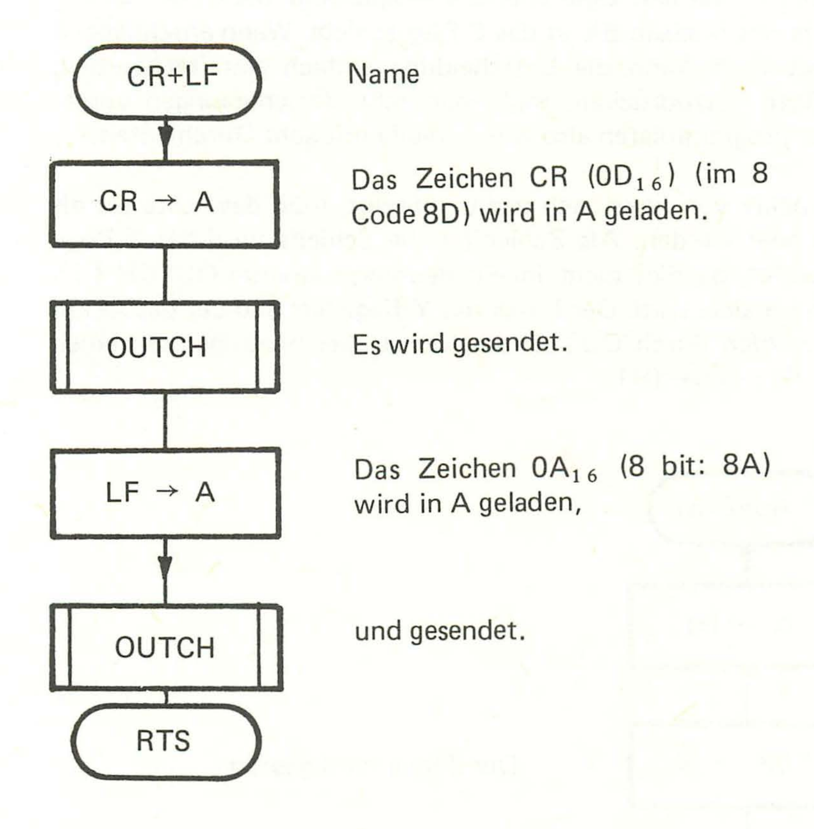
```
USER  LDA   ZP  H
      JMP   AB  00 02
```

Die Unterprogramme:

Wir verwenden den 7 bit ASCII-Code, da die meisten TV-Datenstationen mit diesem Code arbeiten. Das 8. Bit ist dann immer Null zu setzen.

1. ASCIIH: Dieses Unterprogramm wurde in diesem Kapitel schon besprochen.

2. CR+LF:



Der Assemblercode:

```
CR+LF   LDA   IM    0D
        JSR   AB    OUTCH
        LDA   IM    0A
        JSR   AB    OUTCH
        RTS   IMP
```

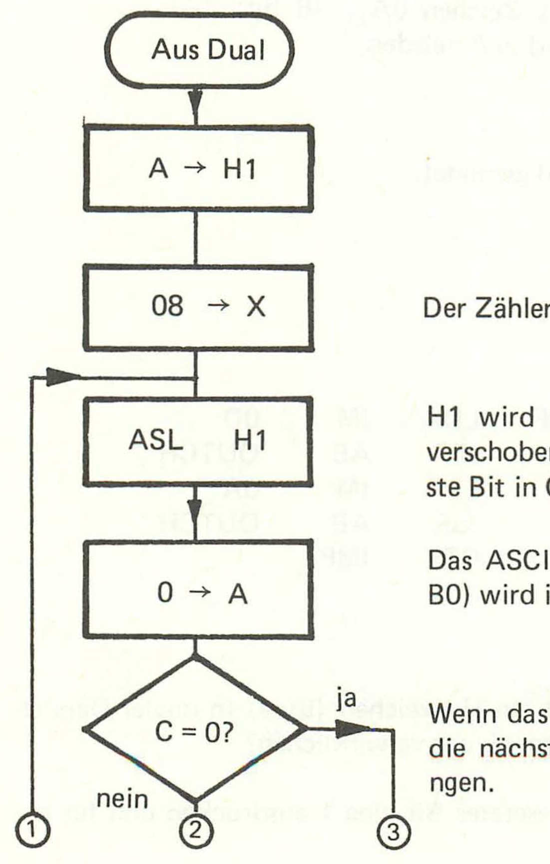
3. Aus Dual:

Dieses Unterprogramm soll ein Hexzeichen (Byte) in dualer Darstellung ausdrucken. Wie können wir das verwirklichen?

Der Drucker soll für ein gesetztes Bit eine 1 ausdrucken und für ein rückgesetztes Bit eine 0.

Die Entscheidung, ob ein Bit gesetzt ist, kann auf unterschiedlichste Art ausgeführt werden. Eine einfache Möglichkeit bietet der Befehl ASL, der ja das höchste Bit in das C-Flag schiebt. Wenn anschließend C abgefragt wird, kann die Entscheidung einfach gestaltet werden. Um ein Byte auszudrucken, muß man acht Verschiebungen durchführen. Wir programmieren also eine Schleife mit acht Durchläufen.

Da der Drucker von links nach rechts schreibt, muß das achte Bit als erstes gesendet werden. Als Zähler für die Schleife wird das X-Register verwendet, da dies nicht im Sendeunterprogramm OUTCH (im Monitor) verändert wird. Der Inhalt des Y-Registers und der des Akkumulators werden durch OUTCH zerstört. Daher benötigen wir einen zweiten Hilfsspeicher (H1).

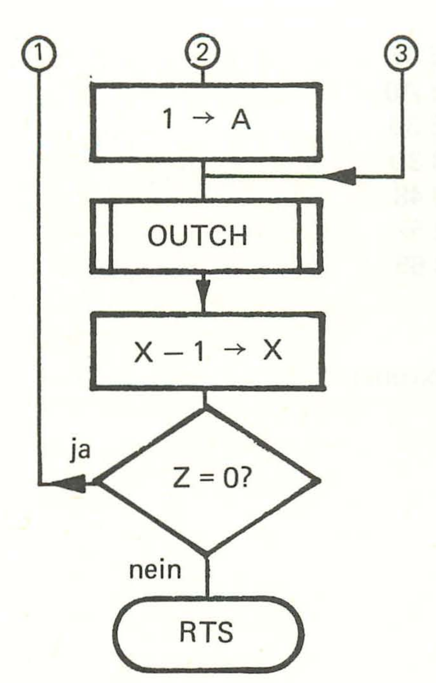


Der Zähler wird gesetzt.

H1 wird um ein Bit nach links verschoben und damit das oberste Bit in C gebracht.

Das ASCII-Zeichen 30_{16} (8 bit: B0) wird in A geladen.

Wenn das C-Flag gesetzt ist, wird die nächste Anweisung übersprungen.



Das ASCII-Zeichen 31_{16} (8 bit: B1) wird in A geladen.

Aus Dual im Assemblercode:

Aus Dual	STA	ZP	H1
	LDX	IM	08
M1	ASL	ZP	H1
	LDA	IM	30
	BCC	R	M2
	LDA	IM	31
M2	JSR	AB	OUTCH
	DEX	IMP	
	BNE	R	M1
	RTS	IMP	

Das ganze Programm im Maschinencode:

Die Adressen sind:	H	00 00
	H1	00 01
	GETCH	1E 5A

OUTSP	1E 9E
OUTCH	1E A0
PRTBYT	1E 3B
CR+LF	03 3D
ASCIIH	03 48
Aus Dual	03 52
USER	03 65

Das Programm im Assemblercode (Hexcode):

```
0300  20  jsr abs 3d  03
0303  20  jsr abs 5a  1e
0306  20  jsr abs 48  03
0309  0a  asl ac
030a  0a  asl ac
030b  0a  asl ac
030c  0a  asl ac
030d  85  sta zp  00
030f  20  jsr abs 5a  1e
0312  20  jsr abs 48  03
0315  05  ora zp  00
0317  85  sta zp  00
0319  20  jsr abs 9e  1e
031c  20  jsr abs 9e  1e
031f  a5  lda zp  00
0321  20  jsr abs 52  03
0324  20  jsr abs 3d  03
0327  20  jsr abs 65  03
032a  85  sta zp  00
032c  20  jsr abs 3b  1e
032f  20  jsr abs 9e  1e
0332  20  jsr abs 9e  1e
0335  a5  lda zp  00
0337  20  jsr abs 52  03
033a  4c  jmp abs 00  03
033d  a9  lda im  0d
033f  20  jsr abs a0  1e
0342  a9  lda im  0a
0344  20  jsr abs a0  1e
0347  60  rts imp
0348  c9  cmp im  41
034a  30  bmi r   03
```

```
034c  18  clc imp
034d  69  adc im  09
034f  29  and im  0f
0351  60  rts imp
0352  85  sta zp  01
0354  a2  ldx im  08
0356  06  asl zp  01
0358  a9  lda im  30
035a  90  bcc r   02
035c  a9  lda im  31
035e  20  jsr abs a0  1e
0361  ca  dex imp
0362  d0  bne r   f2
0364  60  rts imp
0365  a5  lda zp  00
0367  4c  jmp abs 00  02
```

Der Speicherauszug:

```
0300 20 3d 03 20 5a 1e 20 48 03 0a 0a 0a 0a 85 00 20
0310 5a 1e 20 48 03 05 00 85 00 20 9e 1e 20 9e 1e a5
0320 00 20 52 03 20 3d 03 20 65 03 85 00 20 3b 1e 20
0330 9e 1e 20 9e 1e a5 00 20 52 03 4c 00 03 a9 0d 20
0340 a0 1e a9 0a 20 a0 1e 60 c9 41 30 03 18 69 09 29
0350 0f 60 85 01 a2 08 06 01 a9 30 90 02 a9 31 20 a0
0360 1e ca d0 f2 60 a5 00 4c 00 02
```

4.2 Einfache Spiele

In diesem Kapitel werden zwei einfache Spielprogramme vorgestellt, die einen geringen Aufwand an Hardware erfordern.

Die in Kapitel 4.1 gewonnenen Erfahrungen erleichtern uns das Verständnis dieser Programme wesentlich.

1. Ein elektronischer Würfel

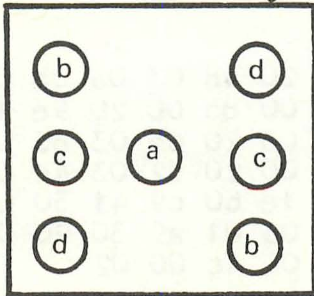
Die Problemstellung:
Sechs Leuchtdioden werden in der Anordnung der sechs Würfelaugen auf eine Experimentierplatte montiert.

Drückt der Spieler auf einen Taster, so soll eine Zufallszahl zwischen 1 und 6 erzeugt werden, die als entsprechendes „Würfelaugenmuster" an dem LED-Feld ablesbar ist.

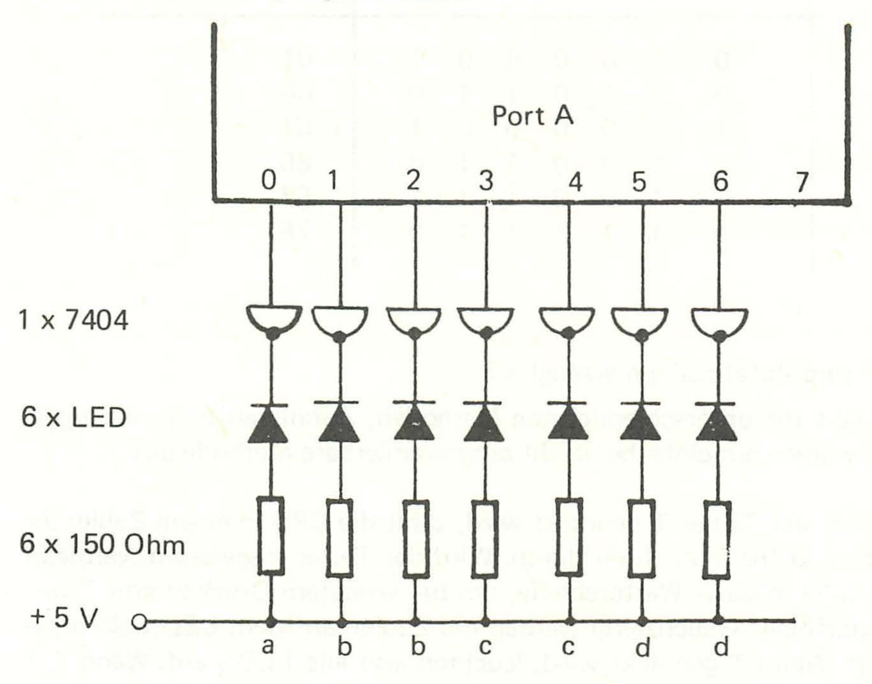
Die Problemlösung:

Erst wird die einfache zusätzliche Hardware besprochen.
Die meisten (in Kapitel 1 vorgestellten) Microcomputer besitzen mindestens zwei bidirektionale Ein- und Ausgabeports (wie z. B. der KIM). Steht dem Leser ein Computer mit nur einem Port zur Verfügung, so wird es ihm nicht schwer fallen, wenn er dieses Programmbeispiel durchgearbeitet hat, die Hardware an nur diesen einen Port anzuschließen. Er muß das Programm nur leicht abändern, was zum Abschluß kurz erläutert wird.

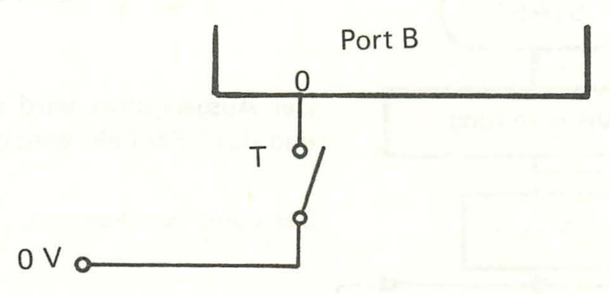
Die 6 Leuchtdioden seien wie folgt angeordnet und bezeichnet:



Ihre Treiber (z. B. 1 x 7404) werden an den Port A, der auf Ausgang zu programmieren ist, angeschlossen.



Den Drucktaster T schließen wir an Bit 0 von Port B an, den wir auf Eingang programmieren.



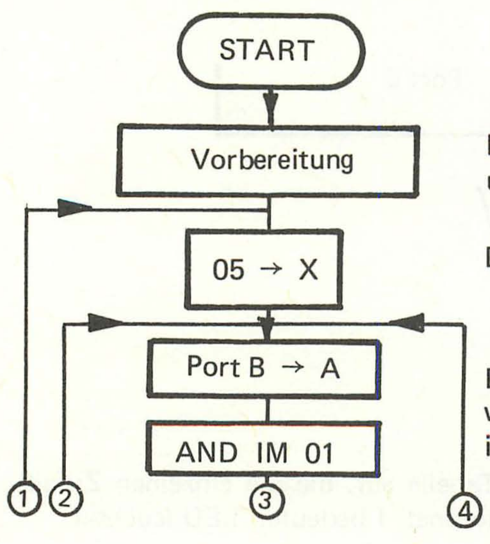
Als nächstes stellen wir eine Tabelle auf, die die einzelnen Zufallszahlen 1 bis 6 dem LED-Feld zuordnet. 1 bedeutet, LED leuchtet.

Zahl	Bit 6 d	5 d	4 c	3 c	2 b	1 b	0 a	Feld (hex)
1	0	0	0	0	0	0	1	01
2	0	0	0	0	1	1	0	06
3	1	1	0	0	0	0	1	61
4	1	1	0	0	1	1	0	66
5	1	1	0	0	1	1	1	67
6	1	1	1	1	1	1	0	7E

Wie sind Zufallszahlen erzeugbar?

Es gibt die unterschiedlichsten Methoden, Zufallszahlen zu erzeugen. Wir wählen ein einfache, leicht programmierbare Methode aus.

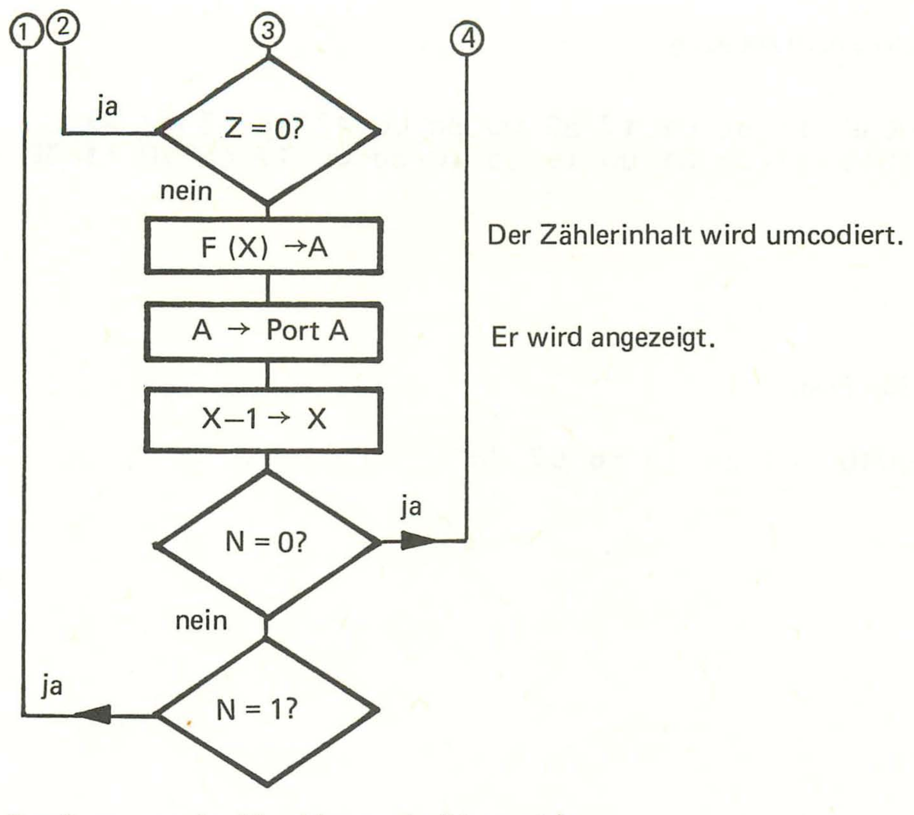
Sobald der Taster T gedrückt wird, zählt die CPU in einem Zähler die Zahlen 0 bis 5 zyklisch durch. Wird der Taster losgelassen, verbleibt die CPU in einer Warteschleife, um bei erneutem Drücken von T weiterzuzählen. Gleichzeitig werden die Zahlen auf dem LED-Feld angezeigt. Wenn T gedrückt wird, leuchten also alle LED's auf. Wenn T in den Ruhezustand zurückgeht, leuchtet ein bestimmtes Würfelmuster auf.



Der Ausgangsport wird definiert und das LED-Feld wird gelöscht.

Der Zähler wird gesetzt.

Der Taster wird abgefragt und, wenn T offen ist, bleibt die CPU in einer Warteschleife.



Das Programm im Maschinencode (Hexcode):

Das Datenfeld wurde in den RAM-Bereich 0010 bis 0016 angelegt.

```
0200  a9  lda im  ff
0202  8d  sta abs 01  17
0205  a9  lda im  00
0207  8d  sta abs 00  17
020a  8d  sta abs 03  17
020d  a2  ldx im  05
020f  ad  lda abs 02  17
0212  29  and im  01
0214  d0  bne r   f9
0216  b5  lda zpx 10
0218  8d  sta abs 00  17
021b  ca  dex imp
021c  10  bpl r   f1
021e  30  bmi r   ec
```

Der Speicherauszug:

```
0200 a9 ff 8d 01 17 a9 00 8d 00 17 8d 03 17 a2 05 ad
0210 02 17 29 01 d0 f9 b5 10 8d 00 17 ca 10 f1 30 ec
```

Das Feld:

```
0010  01 28 15 66 67 7e
```

2. Die Hasenjagd

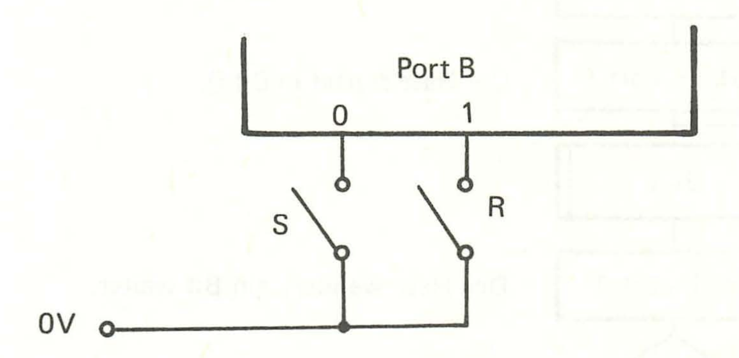
Die Problemstellung:

Ein Hase (Leuchtpunkt) läuft über ein 8 bit-LED-Feld. Trifft der Schuß ihn auf Feld 6, so verlöscht die Anzeige. Andernfalls läuft der Hase zyklisch weiter. War der Schuß ein Treffer, kann man durch Drücken der R-Taste (Reset) das Spiel wieder starten.

Die Problemanalyse:

Die Hardware für dieses Beispiel ist denkbar einfach. Wir schließen an Port A ein 8 bit LED-Feld an, wie im 1. Beispiel beschrieben. Als Treiber könnte man z. B. zwei 7400 Bausteine verwenden.

Die beiden Taster S (Schuß) und R (Reset) werden an Bit 0 und 1 von Port B angeschlossen.



Der Hase soll zyklisch durch das Feld laufen. Wenn er das Feldende bei Bit 7 erreicht, soll er in Bit 0 wieder starten. Es muß eine 1 zyklisch durch das Ausgaberegister geschoben werden. Dies führt der Befehl ROL aus. Dieser Befehl schiebt aber das C-Flag mit durch. Man muß also zu Beginn des Programms das C-Flag löschen.

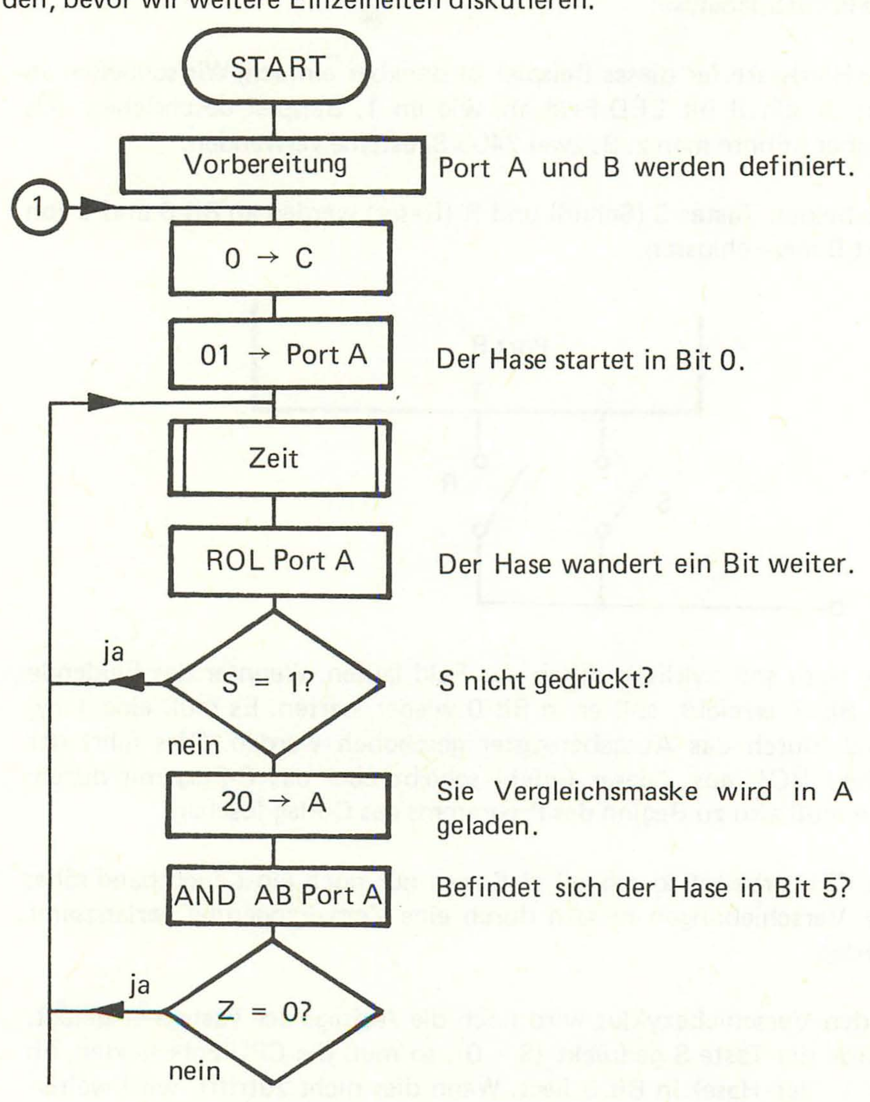
Die CPU arbeitet so schnell, daß man nur noch ein Leuchtband sähe. Die Verschiebungen müssen durch eine Zeitverzögerung verlangsamt werden.

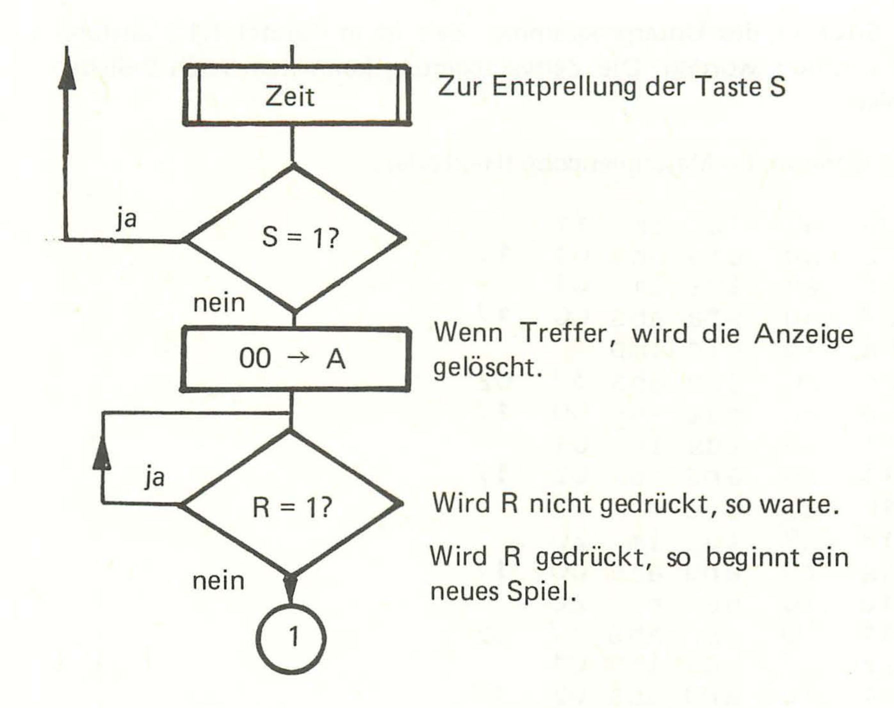
In den Verschiebezyklus wird noch die Abfrage der Taste S eingefügt. Wurde die Taste S gedrückt (S = 0), so muß die CPU entscheiden, ob die 1 (der Hase) in Bit 5 liegt. Wenn dies nicht zutrifft, wird weiter-

geschoben. Trifft es zu, so wird die Anzeige gelöscht und die CPU wartet, bis die Taste R (R = 0) gedrückt wird, um mit einem neuen Spiel zu beginnen.

Die Problemlösung:

Diese Vorgänge sollen nun in einem Ablaufplan veranschaulicht werden, bevor wir weitere Einzelheiten diskutieren.





Die Programmierung einzelner Planelemente muß noch genauer betrachtet werden.

In der Vorbereitungsphase muß der Port A auf Ausgang geschaltet werden.Das Betriebssystem des KIM schaltet Port B automatisch auf Eingang.

Die Abfrage S = 1 kann man so programmieren:

```
     LDA    IM    01
     AND    AB    02 17   Adresse von Port B
     BEQ    R     M       wenn S = 1, dann springe
```

Die Abfrage R = 1 wird genauso programmiert:

```
M3   LDA    IM    02
     AND    AB    02 17
     BNE    R     M3
     BEQ    R     Anfang
```

Die Struktur des Unterprogramms Zeit ist in Kapitel 4.1.2 ausführlich erläutert worden. Die Zeitverzögerung kann man nach Belieben wählen.

Das Programm im Maschinencode (Hexcode):

```
0200  a9  lda im   ff
0202  8d  sta abs 01   17
0205  a9  lda im   01
0207  8d  sta abs 00   17
020a  18  clc imp
020b  20  jsr abs 37   02
020e  2e  rol abs 00   17
0211  a9  lda im   01
0213  2d  and abs 02   17
0216  d0  bne r    f3
0218  a9  lda im   20
021a  2d  and abs 00   17
021d  f0  beq r    ec
021f  20  jsr abs 37   02
0222  a9  lda im   01
0224  2d  and abs 02   17
0227  d0  bne r    e5
0229  a9  lda im   00
022b  8d  sta abs 00   17
022e  a9  lda im   02
0230  2d  and abs 02   17
0233  d0  bne r    f9
0235  f0  beq r    ce
0237  a0  ldy im   a0
0239  a2  ldx im   ff
023b  ca  dex imp
023c  d0  bne r    fd
023e  88  dey imp
023f  d0  bne r    f8
0241  60  rts imp
```

Der Speicherauszug:

```
0200 a9 ff 8d 01 17 a9 01 8d 00 17 18 20 37 02 2e 00
0210 17 a9 01 2d 02 17 d0 f3 a9 20 2d 00 17 f0 ec 20
0220 37 02 a9 01 2d 02 17 d0 e5 a9 00 8d 00 17 a9 02
0230 2d 02 17 d0 f9 f0 ce a0 a0 a2 ff ca d0 fd 88 d0
0240 f8 60
```

4.3 Eine elektronische Orgel

In den elektronischen Musikinstrumenten ist der Microprozessor ein wichtiges Bauelement geworden. Es übernimmt meist die Aufgaben eines Steuerungselementes, das die verschiedenen Klangformungen steuert. Aber auch als Mutteroszillator ist er einsetzbar.

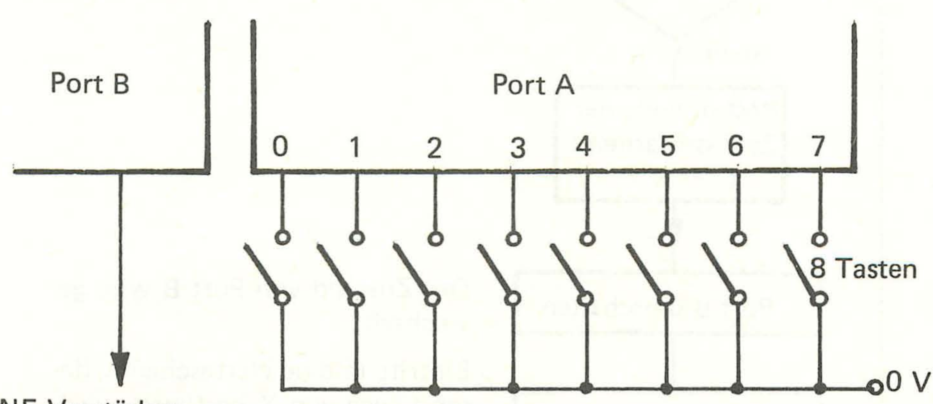
In diesem Kapitel wollen wir unseren Microcomputer als einfache Orgel einsetzen.

Die Problemstellung:

Unsere Orgel ist ein Rechteckgenerator, dessen Frequenz einstellbar sein soll. Sie soll acht Töne (zum Beispiel eine Durtonleiter) erzeugen. Wird eine von acht Tasten gedrückt, erklingt der entsprechende Ton, wird keine Taste gedrückt, erklingt kein Ton. Drückt man zufällig zwei Tasten gleichzeitig, so soll der höhere Ton erklingen.

Die Problemanalyse:

Die notwendige Hardware beschränkt sich auf acht Tasten, die an einen Eingangsport (Port A) angeschlossen werden, und auf einen einfachen NF-Verstärker, der an Bit 0 eines Ausgangsports (Port B) gekoppelt wird.



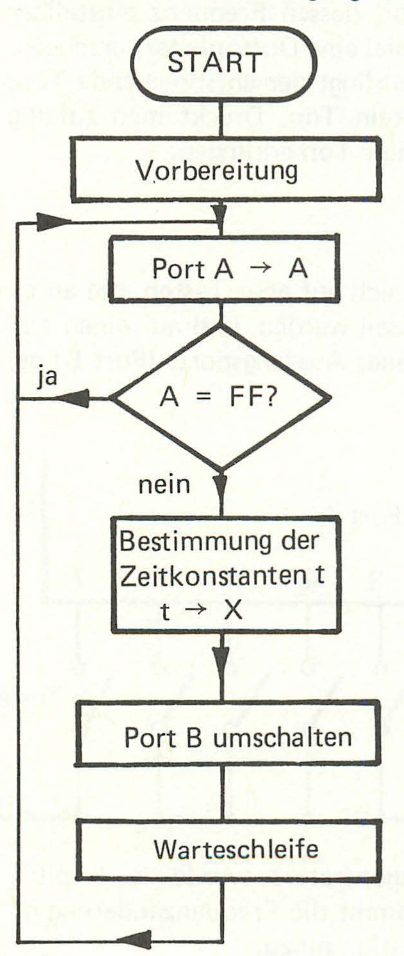
Die Programmierung eines Rechteckgenerators wurde in Kapitel 4.1.2 vorgestellt. Als neues Problem kommt die Frequenzänderung in Abhängigkeit von der Tastenfeldinformation hinzu.

Das die Frequenz bestimmende Element des Generatorprogramms ist die Warteschleife.
Bevor die CPU in die Warteschleife einsteigt, muß sie die Information des Tastenfeldes lesen. Anschließend wird die, zu dieser Information gehörende Zeitkonstante bestimmt. Die Zeitkonstante legt die Länge der Warteschleife und damit die Frequenz des Tones fest.

Wird keine Taste gedrückt, muß die CPU warten, bis eine solche betätigt wird. Sie befindet sich also in einer Tastenabfrageschleife. Es erklingt kein Ton.

Die Problemlösung:

Wir können uns die Vorgänge an einem Ablaufplan veranschaulichen.



Port B muß als Ausgang definiert werden.

Die Information des Tastenfeldes wird gelesen.

Wird keine Taste gedrückt, so bleibe in einer Warteschleife.

Der Zustand von Port B wird gewechselt.

Eintritt in eine Warteschleife, deren Länge von X bestimmt wird. Jetzt ist ungefähr eine halbe Schwingungsdauer verstrichen und es folgt die nächste Hälfte.

Es wird eine Schwingungsperiode in zwei Hälften zerlegt. Am Anfang einer jeden Hälfte bestimmt die CPU die Information des Tastenfeldes.

Jetzt wollen wir uns die einzelnen Programmelemente genauer ansehen.

Die Vorbereitung:

Die Vorbereitung ist ein einfacher Ladevorgang. Das Byte 01_{16} wird in das Datenrichtungsregister von Port B geladen.

```
LDA   IM   01
STA   AB   PBDD
```

Die Leseschleife:

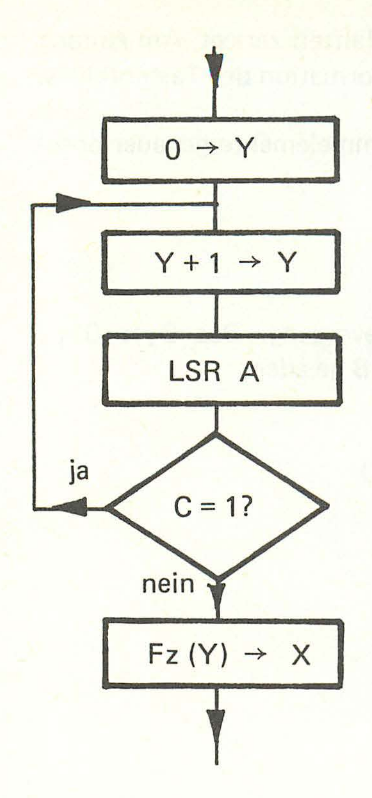
```
M1   LDA   AB   PAD
     CMP   IM   FF
     BEQ   R    M1
```

Die Bestimmung der Zeitkonstanten:

Es erscheint sinnvoll, da wir unsere Orgel beliebig stimmen wollen, die Zeitkonstanten eines jeden Tones in ein Datenfeld Fz abzulegen. Das Datenfeld umfaßt also 8 Byte.

Ist die Tastenfeldinformation in A geladen, so muß die CPU abzählen, welches Bit Null ist. Die Nummer dieses Bits ist die Adresse des zugehörigen Datenbytes in Fz.

Der Abzählvorgang wird am günstigsten durch eine mehrfache Verschiebeoperation mit gleichzeitigem Abzählen durchgeführt.



Das Y-Register wird als Zähler verwendet.

Der Inhalt von A wird 1 bit nach rechts verschoben und das unterste Bit in C gespeichert.

Wenn C = 0, ist der Ton gefunden. Im Y-Register befindet sich die Nummer der gedrückten Taste.

Das Datenbyte, dessen relative Adresse in Y steht, wird in das X-Register geladen.

Im Assemblercode:

```
    LDY  IM   00
M2  INY  IMP
    LSR  AC
    BCS  R    M2
    LDX  ABY  FF   01
```

Als Ladebefehl für das X-Register wird LDX mit absolut Y indizierter Adressierung verwendet. Das 1. Byte von Fz befindet sich in 0200. Daher muß als Anfangsadresse 01FF eingesetzt werden.

Die Warteschleife:

Die Warteschleife ist eine Doppelschleife, bei der die Register X und Y als Zähler benutzt werden. X bestimmte die Länge der Zeitkonstanten.

M3	LDY	IM	0F	0F ist eine experimentell bestimmte Konstante.
M4	DEY	IMP		
	BPL	R	M4	
	DEX	IMP		
	BPL	R	M3	

Das Programm im Maschinençode (Hexcode):

Adressen:		
	PAD	17 00
	PBD	17 02
	PBDD	17 02
	Fz	02 00 bis 02 07

Ein Beispiel für die Zeitkonstanten in Fz:

```
0200   00 01 02 03 04 05 06 07
```

Das Programm im Maschinencode (Hexcode):

```
0208  a9  lda im  01
020a  8d  sta abs 03  17
020d  ad  lda abs 00  17
0210  c9  cmp im  ff
0212  f0  beq r   f9
0214  18  clc imp
0215  a0  ldy im  00
0217  c8  iny imp
0218  4a  lsr ac
0219  b0  bcs r   fc
021b  be  ldx aby ff  01
021e  ee  inc abs 02  17
0221  a0  ldy im  0f
0223  88  dey imp
0224  10  bpl r   fd
0226  ca  dex imp
0227  10  bpl r   f8
0229  4c  jmp abs 0d  02
```

Der Speicherauszug:

```
0208 a9 01 8d 03 17 ad 00 17 c9 ff f0 f9 18 a0 00 c8
0218 4a b0 fc be ff 01 ee 02 17 a0 0f 88 10 fd ca 10
0228 f8 4c 0d 02
```

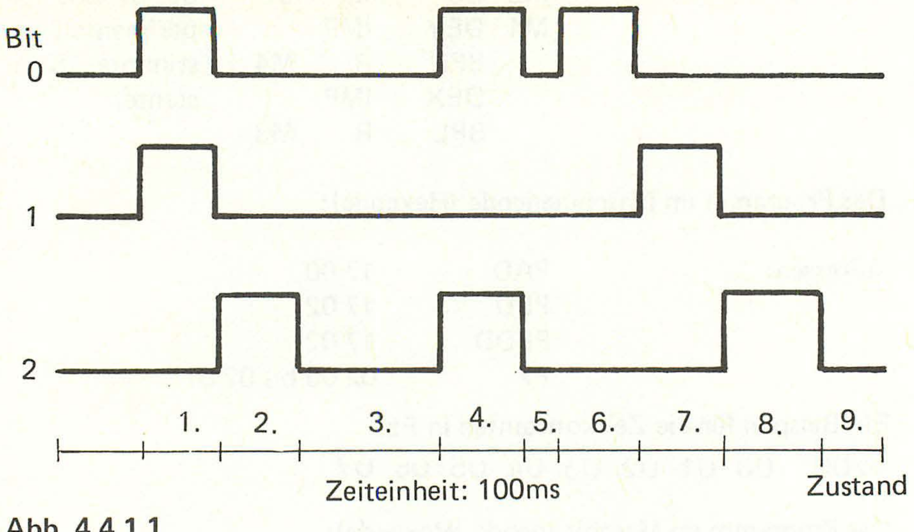


Abb. 4.4.1.1

Abb. 4.4.1.2: Die Schaltzustände

Feld	Zustand	hex	Zeiteinheiten
1	0011	03	2
2	0100	04	2
3	0000	00	6
4	0101	05	2
5	0000	00	1
6	0001	01	2
7	0010	02	2
8	0100	04	2
9	0000	00	4

4.4 Eine Ampelanlage

In diesem Kapitel soll gezeigt werden, wie man einen Microcomputer als Steuerungsanlage programmiert.

Bevor wir uns der Realisierung einer Ampelsteuerung zuwenden, wollen wir, nachdem wir nun schon einige Programme und Programmierkniffe kennenlernten, einen mehrstelligen Funktionsgenerator programmieren.

1. Ein mehrstelliger Funktionsgenerator

In Kapitel 4.1.2 lernten wir zwei einfache Programme zur Erzeugung von Rechteckimpulsen kennen. In den meisten Anwendungen wird aber nicht nur ein periodisches Rechtecksignal verlangt, sondern oft sehr viele, die meist streng synchronisiert sein sollen.

Einen Generator, der mehrere periodische Rechtecksignale erzeugt, die streng synchronisiert sind, nennen wir mehrstelligen Funktionsgenerator. Ein Zweiphasentaktgenerator ist zum Beispiel ein zweistelliger Funktionsgenerator.

Die Problemstellung:

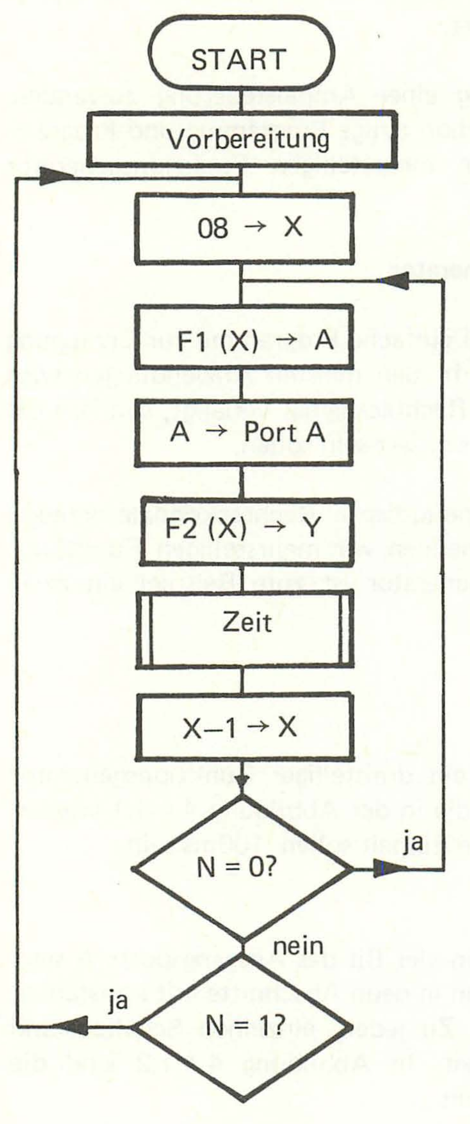
Für eine Ablaufsteuerung wird ein dreistelliger Funktionsgenerator benötigt. Die drei Kanäle sollen die in der Abbildung 4.4.1.1 wiedergegebenen Funktion erzeugen. Die Einheit sollen 100ms sein.

Die Problemlösung:

Die drei Kanäle sollen die unteren vier Bit des Ausgangsports A sein. Das Diagramm (Abb. 4.4.1.1) kann in neun Abschnitte mit konstanten Schaltzuständen zerlegt werden. Zu jedem einzelnen Schaltzustand gehört eine bestimmte Schaltzeit. In Abbildung 4.4.1.2 sind die Schalt- und Zeitzustände angegeben.

Die Schaltzustände werden in einem Feld F1 und die Schaltzeiten in einem Feld F2 abgelegt.

Das Generatorprogramm ist nun leicht verständlich.



X ist der Zustandszähler und der Feldzeiger.

Der durch den Index X adressierte Schaltzustand wird an den Port A gebracht.

Die entsprechende Schaltzeit wird in das Y-Register geladen.

Es folgt das Unterprogramm Zeit

Das Unterprogramm Zeit:

Da relativ lange Verzögerungszeiten erzeugt werden müssen, besteht das Programm aus drei ineinander verschachtelten Schleifen. Das Y-

Register ist der erste Schleifenzähler, die beiden anderen Zähler sind die Datenzellen Z1, Z2, die in der 0. Seite liegen sollen.

Der Zähler Y wurde vom Hauptprogramm mit der Schaltzeitkonstanten geladen. Der Inhalt des Zählers Z1 muß experimentell bestimmt werden. Z2 laden wir mit der größten Zahl (FF_{16}).

M3	LDA	IM	t1
	STA	ZP	Z1
M2	LDA	IM	FF
	STA	ZP	Z2
M1	DEC	ZP	Z2
	BNE	R	M1
	DEC	ZP	Z1
	BNE	R	M2
	DEY	IMP	
	BNE	R	M3
	RTS	IMP	

Das Programm im Maschinencode (Hexcode):

Die Adressen:	F1	00 00 bis 00 08
	F2	00 09 bis 00 11
	Z1	00 12
	Z2	00 13

02	00	A9	LDA	IM	0F	
	02	8D	STA	AB	01 17	(PADD)
	05	A2	LDX	IM	08	
	07	B5	LDA	ZPX	00	
	09	8D	STA	AB	00 17	(PAD)
	0C	B4	LDY	ZPX	09	
	0E	20	JSR	AB	16 02	
	11	CA	DEX	IMP		
	12	10	BPL	R	F5	
	14	30	BMI	R	EF	

Zeit:

02 16	A9	LDA	IM	t1
18	85	STA	ZP	12
1A	A9	LDA	IM	FF
1C	85	STA	ZP	13
1E	C6	DEC	ZP	13
20	D0	BNE	R	FC
22	C6	DEC	ZP	12
24	D0	BNE	R	F4
26	88	DEY	IMP	
27	D0	BNE	R	ED
29	60	RST	IMP	

Der Speicherauszug:

02 00 A9 0F 8D 01 17 A2 08 B5 00 8D 00 17 B4 09 20 16
02 10 02 CA 10 F5 30 EF A9 05 85 12 A9 FF 85 13 C6 13
02 20 D0 FC C6 12 D0 F4 88 D0 ED 60

F1: 00 00 00 04 04 01 00 05 00 04 03
F2: 00 09 04 02 02 02 01 02 06 02 02

Abb. 4.4.2.1

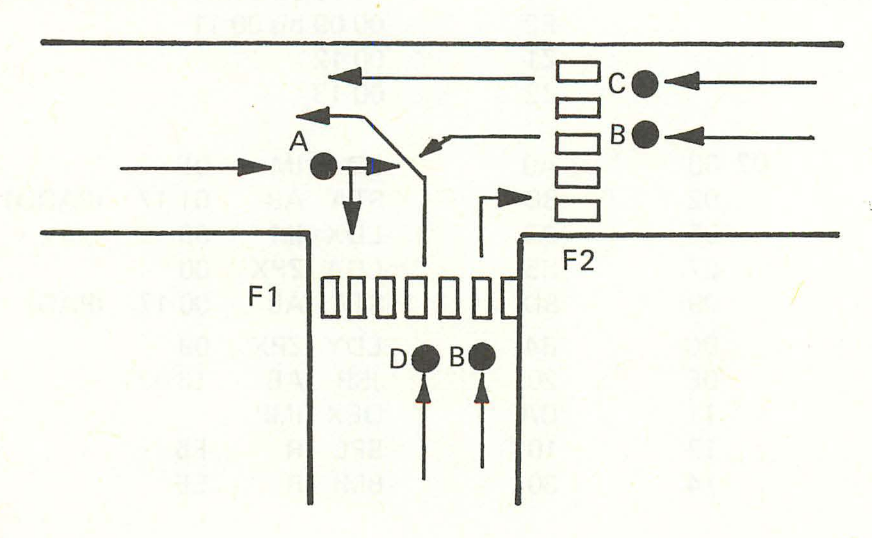
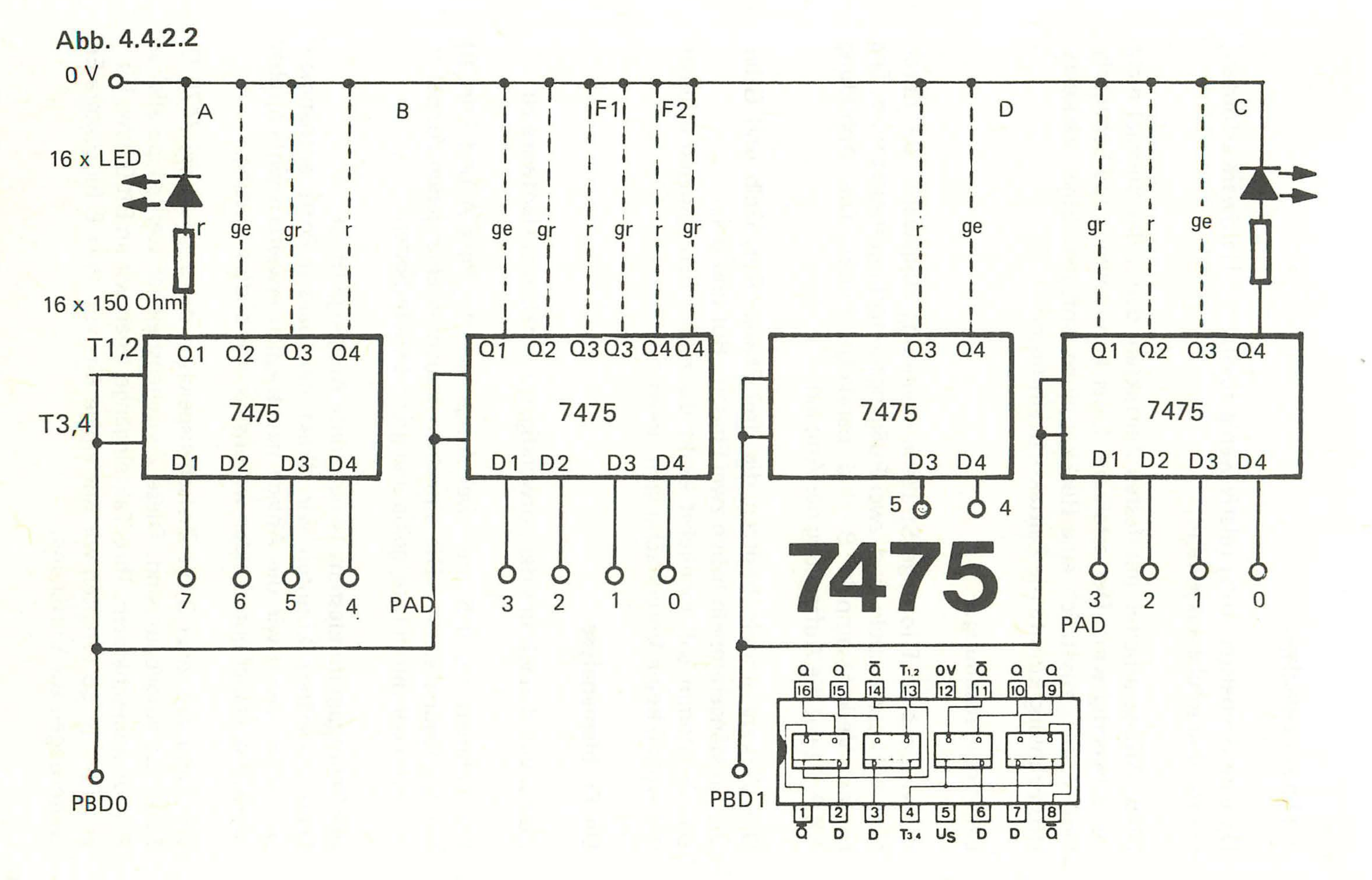


Abb. 4.4.2.2



0 V
A
B
F1
F2
D
C
16 x LED
16 x 150 Ohm
r
ge
gr
T1,2
T3,4
Q1 Q2 Q3 Q4
7475
D1 D2 D3 D4
7 6 5 4
PAD
3 2 1 0
Q1 Q2 Q3 Q3 Q4 Q4
Q3 Q4
D3 D4
5 4
7475
PAD
PBD0
PBD1
Q Q Q̄ T1.2 0V Q̄ Q Q
16 15 14 13 12 11 10 9
1 2 3 4 5 6 7 8
Q̄ D D T3.4 Us D D Q̄

2. Die Ampelanlage

Da eine Ampelsteuerung relativ wenig zusätzliche Hardware erfordert, wurde eine solche ausgewählt.

Diese Ampel arbeitet mit festen Zeittakten, damit das Beispiel nicht zu aufwendig wird. Der erfahrene Leser kann ohne große Schwierigkeiten noch zusätzlich eine Rückkopplung einfügen. Eine Verkehrsstromzählung könnte die Zeittakte beeinflussen.

Die Problemstellung:

Es wurde eine T-förmige Straßeneinmündung ausgewählt, die durch fünf Verkehrsampeln und zwei Fußgängerampeln gesteuert wird. Die beiden Verkehrsampeln B sind parallelgeschaltet. Die Abbildung 4.4.2.1 zeigt die Aufstellung der Ampeln.

Die Verkehrsampeln besitzen die drei Phasen Rot, Gelb und Grün. Die Fußgängerampeln haben zwei Phasen, Rot und Grün.
Das Programm soll möglichst exakt die realen Bedingungen erfüllen. Wir wollen keine Vereinfachungen zulassen.

Die Problemanalyse

Zuerst wenden wir uns der notwendigen zusätzlichen Hardware zu.

Wir nehmen an, daß uns zwei Ausgangsports (Port A und Port B) zur Verfügung stehen. Die einzelnen Leuchten einer jeden Ampel simulieren wir durch rote, gelbe und grüne Leuchtdioden.

Wir benötigen mindestens 14 getrennte Ausgänge (Bits).
Diese Ausgänge könnten wir direkt den beiden Ports entnehmen. Das wäre, wenn wir die Anlage nachträglich erweitern wollen, aber ungünstig. Daher entscheiden wir uns für eine andere Lösung.

Wir benutzen zwei 8 bit Zwischenspeicher, die aus je zwei 7475 D-Flipflops aufgebaut sind. Diese Zwischenspeicher werden parallel an Port A angeschlossen. Ihre Takteingänge legen wir an Bit 0 bzw. Bit 1 von Port B. So erhalten wir uns einige Bits von Port B für spätere Erweiterungen zur Verfügung.

Tabelle 4.4.2.1: Die Schalttabelle der Ampel

Zustand	Zeit	A r g gr	B r g gr	F1	F2	C r g gr	D r g gr
1	t4	0 0 1	1 0 0	1	0	0 0 1	1 0 0
2	t1	0 0 1	1 0 0	0	0	0 0 1	1 0 0
3	t3	0 1 0	1 1 0	0	0	0 0 1	1 0 0
4	t5	1 0 0	0 0 1	0	0	0 0 1	1 0 0
5	t3	1 0 0	0 1 0	0	0	0 1 0	1 1 0
6	t2	1 0 0	1 0 0	0	0	1 0 0	0 0 1
7	t6	1 0 0	1 0 0	0	1	1 0 0	0 0 1
8	t1	1 0 0	1 0 0	0	0	1 0 0	0 0 1
9	t3	1 1 0	1 0 0	0	0	1 1 0	0 1 0
10	t2	0 0 1	1 0 0	0	0	0 0 1	1 0 0

Soll in den ersten Zwischenspeicher eine Information eingeschrieben werden, so müssen wir Bit 0 von Port B auf 1 setzen und die Information an Port A legen. Wird in den zweiten Zwischenspeicher eine Information eingegeben, so müssen von Port B das Bit 0 auf 0 und das Bit 1 auf 1 gesetzt werden. Anschließend kann nach Port A die Information gegeben werden.

Die Abbildung 4.4.2.2 zeigt die entsprechende Schaltung. Für die Fußgängerampeln, die nur Rot- oder Grünzustände einnehmen, reicht ein Bit, da die 7475-Flipflops einen Q- und $\overline{Q}$-Ausgang besitzen.

Bevor wir uns Gedanken über die Entwicklung eines Programmes machen, müssen wir möglichst genau die Funktion der einzelnen Ampeln analysieren.
Das Ergebnis dieser Analyse tragen wir in eine übersichtliche Schalttabelle ein (Tabelle 4.4.2.1).

Da unser Programm möglichst exakt eine reale Situation beschreiben soll, müssen wir alle denkbaren Möglichkeiten an Schaltzuständen einbeziehen. Jeder Schaltzustand besitzt seine eigene Schaltzeit, die wir in die Tabelle unter Zeit eintragen.

Die einzelnen Schaltzustände sollen noch etwas genauer erläutert werden.

Zustand:

1 Grünphase von A und C; Rotphase von B und D; F1 ist grün.
2 F1 wird rot. Die anderen Ampeln bleiben. Die Fußgänger müssen ja erst die Fahrbahn verlassen.
3 Gelbphase
4 Grünphase von B u. C; Rotphase von A und D.
5 Gelbphase
6 Grünphase von D; Rotphase von A, B u. C. Diese Phase ist kurz. Sie soll nur eine Sicherung der Fußgänger bewirken, da in der folgenden Phase die Ampel F2 auf Grün schaltet.
7 Die gleiche Phase wie 6 nur F2 ist grün.
8 F2 wird wieder rot. Diese Phase hat die gleiche Funktion wie die Phase 2.
9 Gelbphase
10 Die Vorbereitungsphase für die 1. Phase

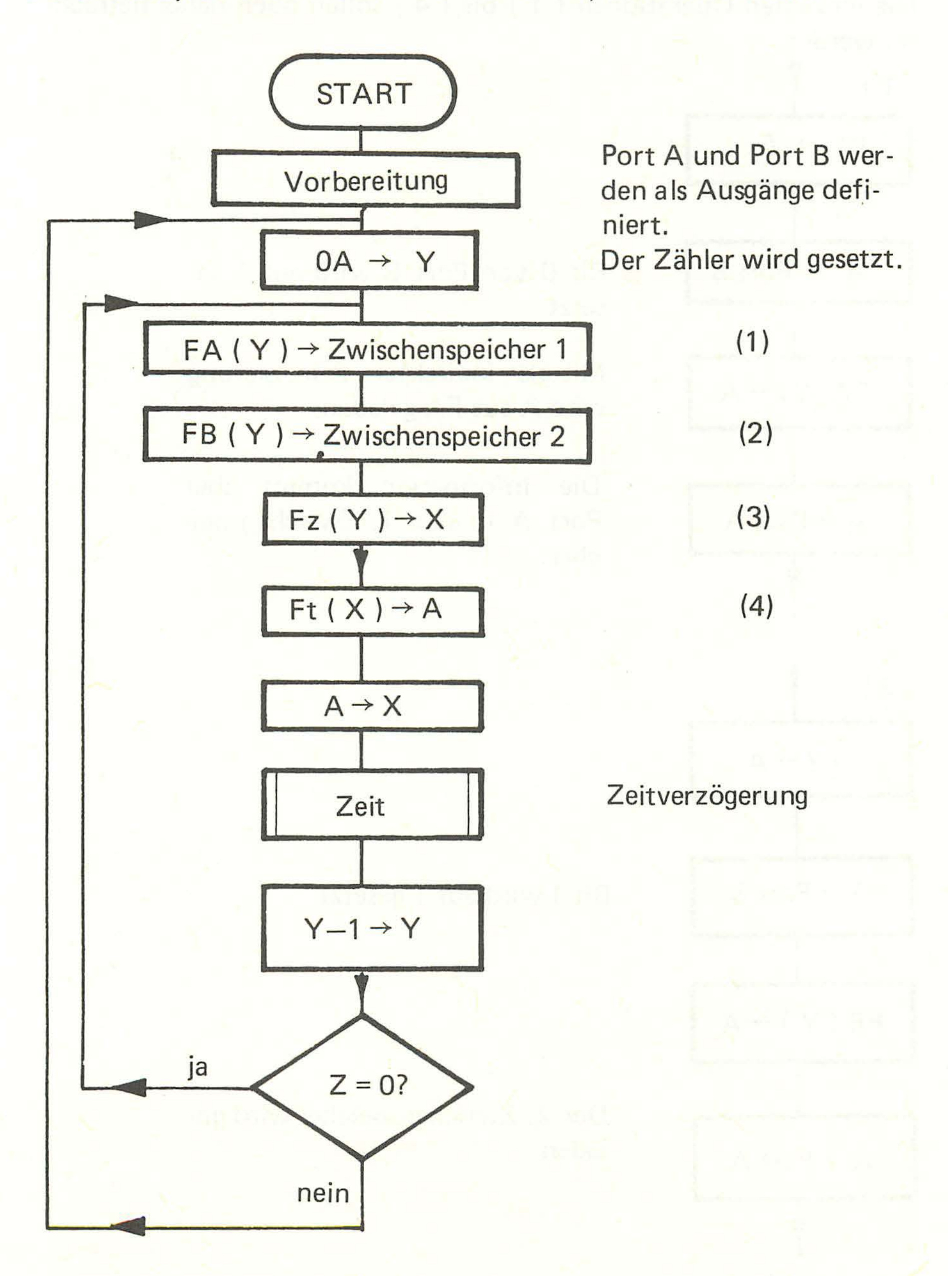
Wir haben also zehn unterschiedliche Schaltzustände mit sehr verschiedenen Schaltzeiten erkannt. Die Schaltzeiten drücken wir durch ein Vielfaches einer Grundzeiteinheit t aus.

Die Problemlösung:

Die Tabelle 4.4.2.1 wird in Form von drei Feldern der Länge 10 byte abgespeichert. Feld FA beinhaltet die Daten für die Ampeln A, B, F1 und F2. Das Feld FB beinhaltet die Daten der Ampeln C und D. Das Feld Fz enthält die Adressen der Zeitvielfachen der Schaltzeiten.

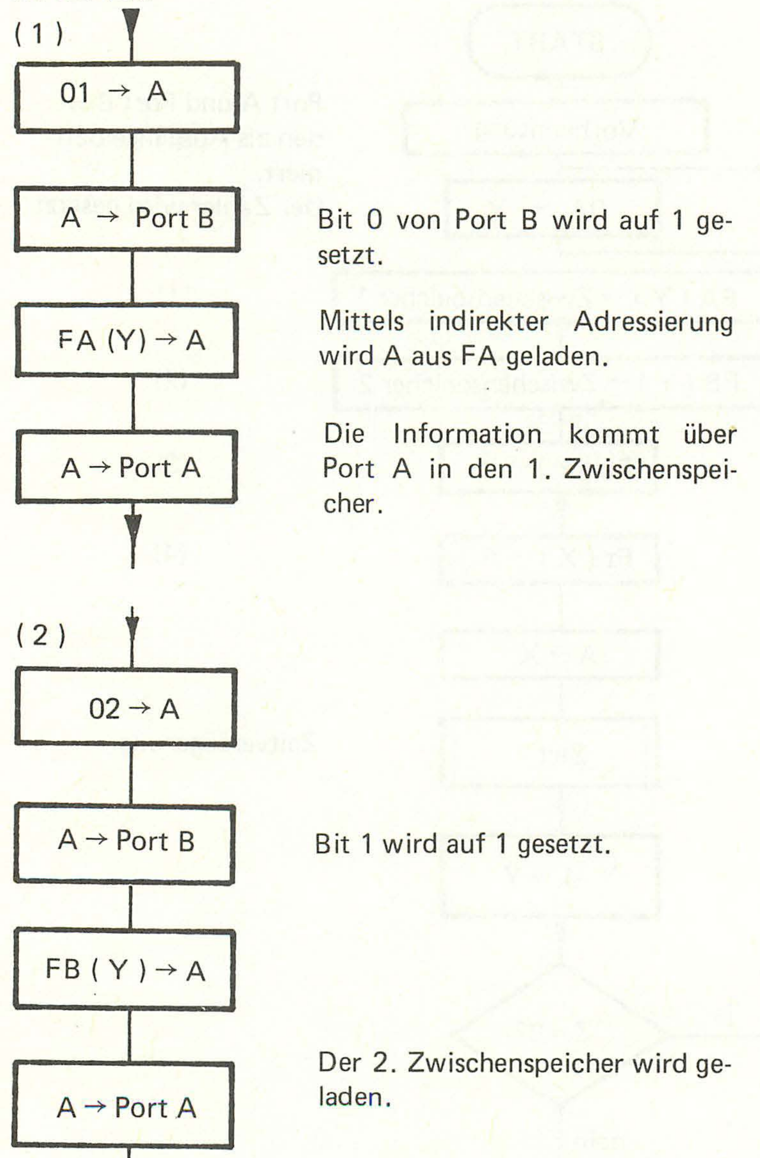
In einem weiteren Feld Ft werden nun die Zeitvielfachen abgespeichert. Wir wählen diesen, etwas umständlich erscheinenden Weg, um bei späteren Änderungen die Zeitvielfachen durch einfache Schreiboperationen per Programm verändern zu können. Sonst wären häufig zwei Schreiboperationen mit relativ komplizierten Adressrechnungen notwendig.
Das Programm ist eine Erweiterung des 1. Beispiels, allerdings mit wesentlichen Änderungen.

Da 10 Schaltzeiten vorliegen, stellt das Programm eine Schleife mit 10 Durchgängen dar, die sich periodisch wiederholen. Als Schleifenzähler wurde das Y-Register ausgewählt, das gleichzeitig ein Zeiger ist, der auf die entsprechenden Daten der Felder FA, FB und Fz weist.



Die Felder FA, FB und Fz werden von oben nach unten angelegt. Das Feld Ft belegen wir von unten nach oben.

Die einzelnen Operationen (1) bis (4) sollen noch näher betrachtet werden.



Bit 0 von Port B wird auf 1 gesetzt.

Mittels indirekter Adressierung wird A aus FA geladen.

Die Information kommt über Port A in den 1. Zwischenspeicher.

Bit 1 wird auf 1 gesetzt.

Der 2. Zwischenspeicher wird geladen.

Im Assemblercode:

```
( 1 )  LDA   IM    01
       STA   AB    02 17
       LDA   ABY   05 02     Adresse  von  FA
       STA   AB    00 17
( 2 )  LDA   IM    02
       STA   AB    02 17
       LDA   ABY   0F 02     Adresse  von  FB
       STA   AB    00 17
( 3 )  LDX   ABY   19 02     Adresse von Fz
( 4 )  LDA   ABX   00 02     Adresse von Ft
```

Das Unterprogramm Zeit ist, da wir große Verzögerungszeiten benötigen, aus vier ineinander verschachtelten Schleifen aufgebaut. Wir verwenden noch drei externe Zähler Z1, Z2 und Z3. Das X-Register ist der vierte Zähler.

Die Zeiteinheit t kann beliebig gewählt werden. t = 2 entspricht etwa 1s.

Der Assemblercode von Zeit:

```
M4   LDA   IM    t
     STA   AB    Z1
M3   LDA   IM    FF
     STA   AB    Z2
M2   LDA   IM    FF
     STA   AB    Z3
M1   DEC   AB    Z3
     BNE   R     M1
     DEC   AB    Z2
     BNE   R     M2
     DEC   AB    Z1
     BNE   R     M3
     DEX   IMP
     BNE   R     M4
     RTS   IMP
```

Das Programm im Maschinencode (Hexcode):

Die Adressen:	Z1	03 00
	Z2	03 01
	Z3	03 02

Das Hauptprogramm:

```
0224   a9   lda im   ff
0226   8d   sta abs 01   17
0229   8d   sta abs 03   17
022c   a0   ldy im   0a
022e   a9   lda im   01
0230   8d   sta abs 02   17
0233   b9   lda aby 05   02
0236   8d   sta abs 00   17
0239   a9   lda im   02
023b   8d   sta abs 02   17
023e   b9   lda aby 0f   02
0241   8d   sta abs 00   17
0244   be   ldx aby 19   02
0247   bd   lda abx 00   02
024a   aa   tax imp
024b   20   jsr abs 03   03
024e   88   dey imp
024f   d0   bne r    dd
0251   4c   jmp abs 2c   02
```

Der Speicherauszug:

```
0224 a9 ff 8d 01 17 8d 03 17 a0 0a a9 01 8d 02 17 b9
0234 05 02 8d 00 17 a9 02 8d 02 17 b9 0f 02 8d 00 17
0244 be 19 02 bd 00 02 aa 20 03 03 88 d0 dd 4c 2c 02
```

Ft:

```
0200   03 01 01 1e 14 14
```

FA:

```
0206   30 d0 90 91 90 88 84 58 30 32
```

FB:

```
0210   0c 32 21 21 21 16 0c 0c 0c 0c
```

Fz:

```
021a   01 02 00 05 01 02 04 02 00 03
```

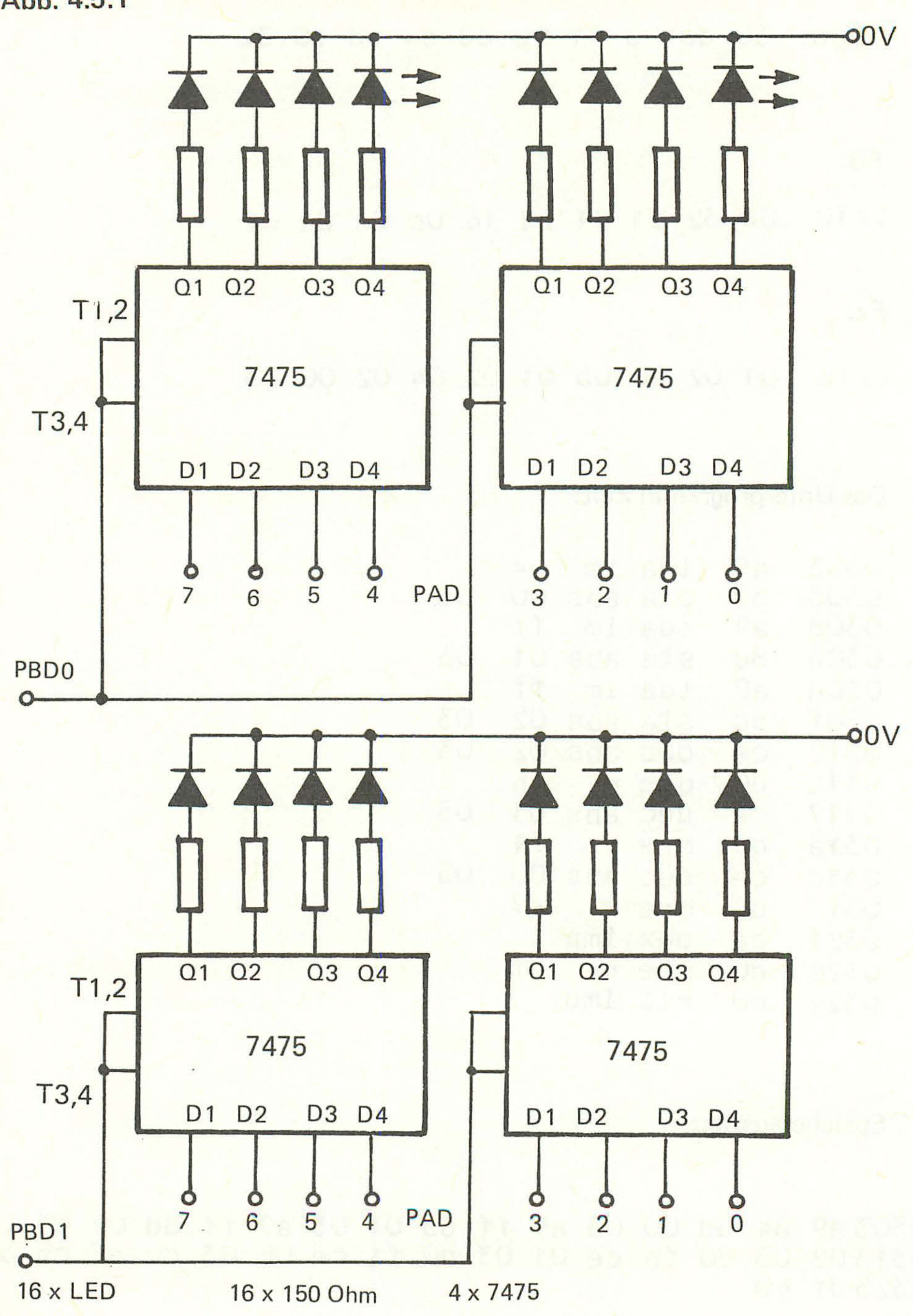
Das Unterprogramm Zeit:

```
0303  a9  lda im  04
0305  8d  sta abs 00  03
0308  a9  lda im  ff
030a  8d  sta abs 01  03
030d  a9  lda im  ff
030f  8d  sta abs 02  03
0312  ce  dec abs 02  03
0315  d0  bne r   fb
0317  ce  dec abs 01  03
031a  d0  bne r   f1
031c  ce  dec abs 00  03
031f  d0  bne r   e7
0321  ca  dex imp
0322  d0  bne r   df
0324  60  rts imp
```

Speicherauszug:

```
0303 a9 04 8d 00 03 a9 ff 8d 01 03 a9 ff 8d 02 03 ce
0313 02 03 d0 fb ce 01 03 d0 f1 ce 00 03 d0 e7 ca d0
0323 df 60
```

Abb. 4.5.1



0V
Q1 Q2 Q3 Q4
T1,2
7475
T3,4
D1 D2 D3 D4
7 6 5 4 PAD
3 2 1 0
PBD0
0V
PBD1
16 x LED
16 x 150 Ohm
4 x 7475

4.5 Programme für eine 2 x 8 bit LED-Anzeige

Welche Anwendungsmöglichkeiten der Programmierer für seinen Microcomputer finden kann, wenn er nur sehr einfache zusätzliche Hardware verwendet, soll Ihnen in diesem Kapitel gezeigt werden.
Für eine einfache doppelte 8 bit LED-Anzeige werden zwei kurze und zwei umfangreiche Programme vorgestellt.

Vielleicht wird der Leser durch diese Programme angeregt, eigene Entwicklungen auszuprobieren? Allein diese LED-Anzeige bietet eine Fülle weiterer Anwendungsmöglicheiten.

Nun die notwendige Hardware!
Wir schließen an einen 8 bit Ausgangsport zwei 8 bit Register (2 x 7475) an. Die Register dienen als Zwischenspeicher. Ihre Ausgänge treiben je eine Leuchtdiode. Die Takteingänge der Register werden an Bit 0 und Bit 1 eines weiteren Ausgangsports unseres Microcomputers gelegt. Ist eines dieser Bits 1, so wird die Information des Datenausgangsports in das betreffende Register übernommen und im LED-Feld angezeigt.
Die LED-Felder sollten untereinander angeordnet sein.
Abbildung 4.5.1 zeigt die Schaltung.

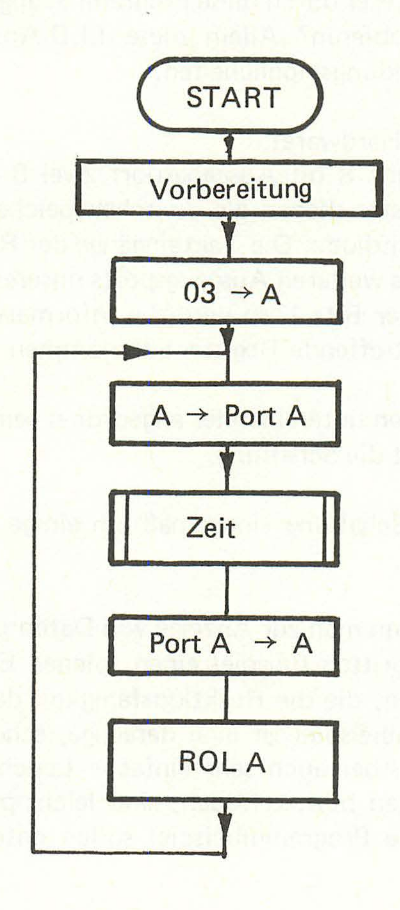
Natürlich kann die Schaltung sinngemäß um einige weitere LED-Felder erweitert werden.

Diese LED-Felder kann man zur Anzeige von Datenregistern einsetzen. Wir besprechen im dritten Beispiel einen solchen Einsatz. Man kann Spiele programmieren, die die Reaktionsfähigkeit des Spielers testen. Das vierte Programmbeispiel ist eine derartige, schon recht umfangreiche Anwendung. Aber auch sehr einfache Leuchteffekte, die den Betrachter überraschen bzw. erfreuen, sind leicht programmiert. Das erste und das zweite Programmbeispiel sollen entsprechende Anregungen geben.

1. Ein Lauflicht

Wir benutzen nur das erste Feld. Ein bestimmtes Bitmuster soll angezeigt werden und dann periodisch durch das Feld von rechts nach links laufen.

Zu Beginn des Programms muß der Programmierer dafür sorgen, daß die Ports auf Ausgang geschaltet werden. Dann wird ein bestimmtes Bitmuster in den Port A geschrieben und eine bestimmte Zeit gewartet. Anschließend muß das Muster um 1 Bit verschoben werden und zur Anzeige kommen. Natürlich folgt die Warteschleife, bevor neu verschoben wird. Das Programm ist als Endlosschleife zu verstehen.



Als Unterprogramm verwenden wir das Unterprogramm Zeit von Kapitel 4.4. Die Zeitkonstante kann in diesem Programm in weiten Grenzen variiert werden.

Das Programm im Maschinencode (Hexcode):

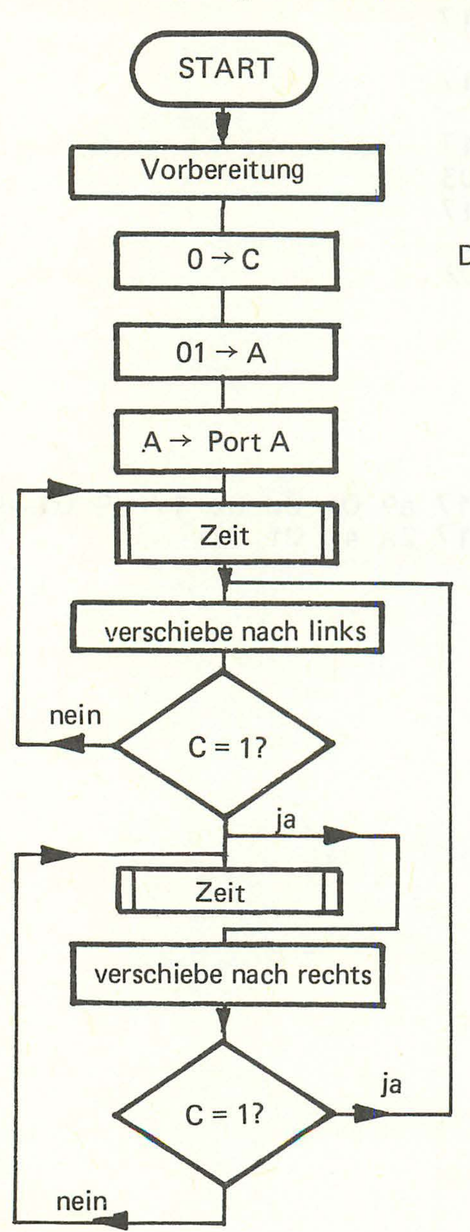
```
0200  a9  lda im  ff
0202  8d  sta abs 01  17
0205  8d  sta abs 03  17
0208  a9  lda im  01
020a  8d  sta abs 02  17
020d  a9  lda im  01
020f  8d  sta abs 00  17
0212  20  jsr abs 03  03
0215  ad  lda abs 00  17
0218  2a  rol ac
0219  4c  jmp abs 0f  02
```

Der Speicherauszug:

```
0200 a9 ff 8d 01 17 8d 03 17 a9 01 8d 02 17 a9 01 8d
0210 00 17 20 03 03 ad 00 17 2a 4c 0f 02
```

2. Ein Lichtpendel

Mit Hilfe der verschiedenen Rotationsbefehle des 6502 kann man leicht ähnliche Programme entwickeln.



Das C-Flag muß gelöscht werden.

Will man zum Beispiel das Bitmuster bis zum linken Feldrand verschieben, um es dann wieder in die andere Richtung zurückzuschieben, so hat man ein " schwingendes Bild" oder "Lichtpendel" programmiert.

Da die Verschiebebefehle des 6502 immer das C-Flag beeinflussen, kann der Programmierer leicht einen Test einbauen, der entscheidet, wann das Bild den linken oder rechten Rand erreicht hat.

Das Programm muß um einen zweiten Teil erweitert werden, in dem die Rechtsverschiebung durchgeführt wird.

Die Verschiebungen werden in diesem Beispiel etwas anders durchgeführt. Dem aufmerksamen Leser wird nicht entgangen sein, daß der Befehl ROL auch eine absolute Adressierung besitzt. Das bedeutet, daß der 6502 auch in einer Speicherzelle ein Bitmuster verschieben kann. Natürlich ist dieser Befehl, genau betrachtet, ein kleines Microprogramm, denn er bewirkt ein Lesen der Speicherzelle mit anschließendem Verschieben und wieder einem Laden.

Wir verwenden nun für die Linksverschiebung den Befehl ROL AB Port A und für die Rechtsverschiebung ROR AB Port A.
Daher entfallen die Lese- und Ladebefehle.

Für die Zeitverzögerung wurde ein etwas einfacheres Programm entwickelt, das ab Adresse 0250 beginnt. Es werden nur zwei Zähler (Adressen: 00 00, 00 01) benutzt.

Das Programm im Maschinencode:

```
0200  a9  lda im  ff
0202  8d  sta abs 01  17
0205  8d  sta abs 03  17
0208  a9  lda im  01
020a  8d  sta abs 02  17
020d  18  clc imp
020e  a9  lda im  01
0210  8d  sta abs 00  17
0213  20  jsr abs 50  02
0216  2e  rol abs 00  17
0219  b0  bcs r   06
```

```
021b  4c  jmp abs 13  02
021e  20  jsr abs 50  02
0221  6e  ror abs 00  17
0224  b0  bcs r   f0
0226  4c  jmp abs 1e  02
```

Der Speicherauszug:

```
0200 a9 ff 8d 01 17 8d 03 17 a9 01 8d 02 17 18 a9 01
0210 8d 00 17 20 50 02 2e 00 17 b0 06 4c 13 02 20 50
0220 02 6e 00 17 b0 f0 4c 1e 02
```

Das Verzögerungsprogramm:

```
0250  a9  lda im  0f
0252  85  sta zp  00
0254  a9  lda im  ff
0256  85  sta zp  01
0258  c6  dec zp  01
025a  d0  bne r   fc
025c  c6  dec zp  00
025e  d0  bne r   f4
0260  60  rts imp
```

Der Speicherauszug:

```
0250 a9 0f 85 00 a9 ff 85 01 c6 01 d0 fc c6 00 d0 f4
0260 60
```

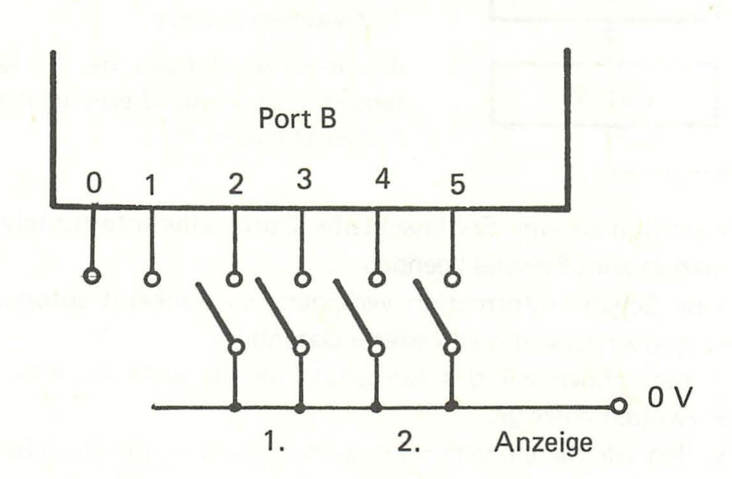
3. Ein Teilprogramm eines Monitors

Wir nehmen an, daß unser LED-Feld ein Bestandteil des Bedienungsfeldes eines Microcomputers ist. Der Monitor, der dieses Bedienungsfeld steuert, hat unter anderem die Aufgabe, die Inhalte bestimmter Speicherzellen über das LED-Feld auszugeben.
Welche Zellen angezeigt werden, bestimmt ein Schalterfeld.

Wir wollen uns nun damit beschäftigen, wie ein solches Teilprogramm eines Monitors aufgebaut wird.

Die Problemstellung:

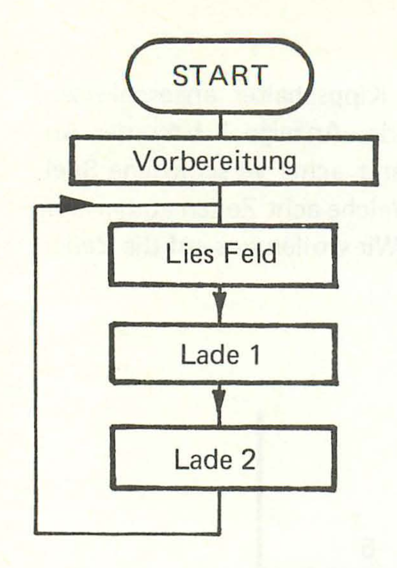
An den Port B werden zusätzlich vier Kippschalter angeschlossen. Je zwei Schalter bestimmen den Inhalt der Anzeige 1 bzw. der Anzeige 2. Wir können also je vier (insgesamt acht) verschiedene Speicherzellen auf dem LED-Feld anzeigen. Welche acht Zellen ausgewählt werden sollen, bestimmt ein Adressfeld. Wir wollen uns auf die Zellen der 0. Speicherseite beschränken.



Die Problemlösung:

Die Adressen der ausgewählten Speicherzellen sollen in einem Adressfeld abgelegt werden, damit man möglichst leicht andere Adressen auswählen kann.

Wir benötigen also zwei Datenfelder mit je 4 byte Umfang. Man kann mit den Schaltern (je Anzeige) 4 verschiedene Dualzahlen setzen 00, 01, 10 und 11. Diese Zahlen lassen sich als Zeiger verstehen, die auf die entsprechende Zelle des zugehörigen Datenfeldes weisen. Das Datenbyte wiederum ist ein Zeiger, der auf die angewählte Speicherzelle zeigt. Der Programmierer muß also zweimal eine indirekte Adressierung programmieren.



Der Inhalt des Schalterfeldes wird gelesen.

Bestimme die Adresse des 1. Datenbytes und lade dieses in den 1. Zwischenspeicher.

Bestimme die Adresse des 2. Datenbytes und lade dieses in den 2. Zwischenspeicher.

Das Programm ist eine Endlosschleife. Durch eine Interrupterzeugung kann man es zum Beispiel beenden.
Wird eine Schalterinformation verändert, so erscheint sofort auf der zugehörigen Anzeige das adressierte Datenbyte.
Mit F1 bezeichnen wir das Adressfeld für die erste Anzeige, mit F2 das der zweiten Anzeige.
Nun wollen wir die einzelnen Programmelemente ausführlicher analysieren.

Die Vorbereitung:

Port A wird als Ausgang definiert. Port B müssen wir nun als Ausgang (Bit 0 und 1) und als Eingang (Bit 2 bis 5) definieren.

```
LDA  IM  03
STA  AB  PBDD
LDA  IM  FF
STA  AB  PADD
```

Lies Feld:

Das Schalterfeld muß gelesen werden. Anschließend sollen die Informationen der beiden Doppelschalter voneinander isoliert und in Dual-

zahlen umgeformt werden.

LDA	AB	Port B	
LSR	AC		Bit 0 und Bit 1 werden heraus-
LSR	AC		geschoben.
AND	IM	0F	Die unteren (wichtigen) Bits werden isoliert.
STA	AB	H	Sie werden in einem Hilfsspeicher zwischengespeichert.
AND	IM	03	Nun werden die Schalterstellungen der Schalter von Anzeige 1 isoliert.
TAY	IMP		Sie werden in das Y-Register transportiert.

Lade 1:

Der Zeiger, der auf das Adressbyte in F1 weist, befindet sich im Y-Register. Mittels der Y-indizierten Adressierung wird das Adressbyte in das X-Register geladen. Nun bringt die CPU das gewünschte Datenbyte mittels der X-indizierten ZP Adressierung in den Akkumulator. Vorher muß aber Bit 0 von Port B gesetzt werden, damit die CPU anschließend das Datenbyte in die Anzeige 1 einschreiben kann.

Um ein Überschreiben auf die Anzeige 2 zu vermeiden, wird Bit 0 von Port B gelöscht. Da die CPU sehr schnell arbeitet, muß auch Port A gelöscht werden. Sonst wäre das Datenbyte auch in Anzeige 2 erkennbar.

```
LDX   ABY   F1
LDA   IM    01
STA   AB    Port B
LDA   ZPX   00
STA   AB    Port A
LDA   IM    00
STA   AB    Port B
STA   AB    Port A
```

Lade 2:

Im Hilfsspeicher H befindet sich die Information des zweiten Schalterfeldes. Durch zweimaliges Rechtsverschieben ist sie isoliert und in eine Dualzahl umgeformt. Diese wird nun in das Y-Register geladen. Das Adressbyte aus F2 wird in das X-Register geholt. Anschließend setzt die CPU Bit 1 von Port B, um die Anzeige 2 anzusprechen. Das betreffende Datenbyte holt die CPU in den Akkumulator, um es dann an den Port A abzugeben. Zum Abschluß werden wieder Port B und Port A gelöscht, um ein Überschreiben auf Anzeige 1 zu vermeiden.

```
LSR  AB   H
LSR  AB   H
LDY  AB   H
LDX  ABY  F2
LDA  IM   02
STA  AB   Port B
LDA  ZPX  00
STA  AB   Port A
LDA  IM   00
STA  AB   Port B
STA  AB   Port A
```

Das Programm im Maschinencode (Hexcode):

Die Adressen:		
	H	03 00
	F1	03 01 bis 03 04
	F2	03 05 bis 03 08
	PADD	1701
	PAD	1700
	PBDD	1703
	PBD	1702

Bei einigen Microcomputern kann der Programmierer die Interruptvektoren beliebig setzen. Dann liegt es nahe, den Vektor für IRQ auf die Adresse 0309 und den von NMI auf die Startadresse des Systemmonitors zu setzen.

Hat man die Vektoren so gesetzt, kann das Programm zum Testen anderer Programme herangezogen werden. Durch einen BRK-Befehl an entsprechender, zu testender Programmstelle, erzwingt man dann einen Sprung in unser Programm. Durch einfaches Schaltereinstellen kann man die gewünschten Speicherzellen anzeigen. Soll weitergearbeitet werden, muß ein NM-Interrupt erzeugt werden.

Das Programm im Maschinencode (Hexcode):

```
0309  a9  lda im  03
030b  8d  sta abs 03  17
030e  a9  lda im  ff
0310  8d  sta abs 01  17
0313  ad  lda abs 02  17
0316  4a  lsr ac
0317  4a  lsr ac
0318  29  and im  0f
031a  8d  sta abs 00  03
031d  29  and im  03
031f  a8  tay imp
0320  be  ldx aby 01  03
0323  a9  lda im  01
0325  8d  sta abs 02  17
0328  b5  lda zpx 00
032a  8d  sta abs 00  17
032d  a9  lda im  00
032f  8d  sta abs 02  17
0332  8d  sta abs 00  17
0335  4e  lsr abs 00  03
0338  4e  lsr abs 00  03
033b  ac  ldy abs 00  03
033e  be  ldx aby 05  03
0341  a9  lda im  02
0343  8d  sta abs 02  17
0346  b5  lda zpx 00
0348  8d  sta abs 00  17
034b  a9  lda im  00
034d  8d  sta abs 02  17
0350  8d  sta abs 00  17
0353  4c  jmp abs 13  03
```

Der Speicherauszug:

```
0309 a9 03 8d 03 17 a9 ff 8d 01 17 ad 02 17 4a 4a 29
0319 0f 8d 00 03 29 03 a8 be 01 03 a9 01 8d 02 17 b5
0329 00 8d 00 17 a9 00 8d 02 17 8d 00 17 4e 00 03 4e
0339 00 03 ac 00 03 be 05 03 a9 02 8d 02 17 b5 00 8d
0349 00 17 a9 00 8d 02 17 8d 00 17 4c 13 03
```

4. Ein Fangspiel

Da wir nun schon einige Programmiererfahrungen haben, können wir uns ruhig auch einmal ein etwas komplizierteres Problem vornehmen.

Unser Microcomputer soll nun als Reaktionsspiel arbeiten.

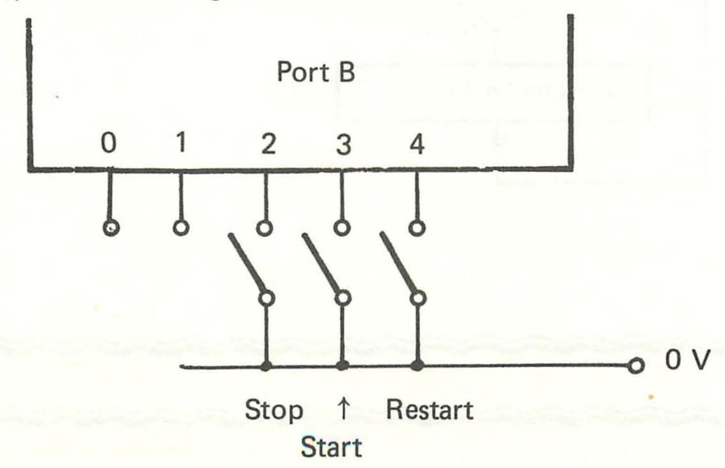
Die Problemstellung:

Drei Leuchtpunkte laufen zyklisch in Anzeige 1 von rechts nach links. Drei weitere Leuchtpunkte laufen zyklisch in Anzeige 2 von links nach rechts. Betätigt der Spieler eine Stop-Taste, so bleiben die Leuchtpunkte eine gewisse Zeit stehen, um dann zu verlöschen. Anschließend wird die Punktzahl, Anzahl der Leuchtpunkte, die übereinander stehen, in Anzeige 2 angezeigt. Anzeige 1 zeigt die Summe aller Punkte aus den vorherigen Durchläufen an.

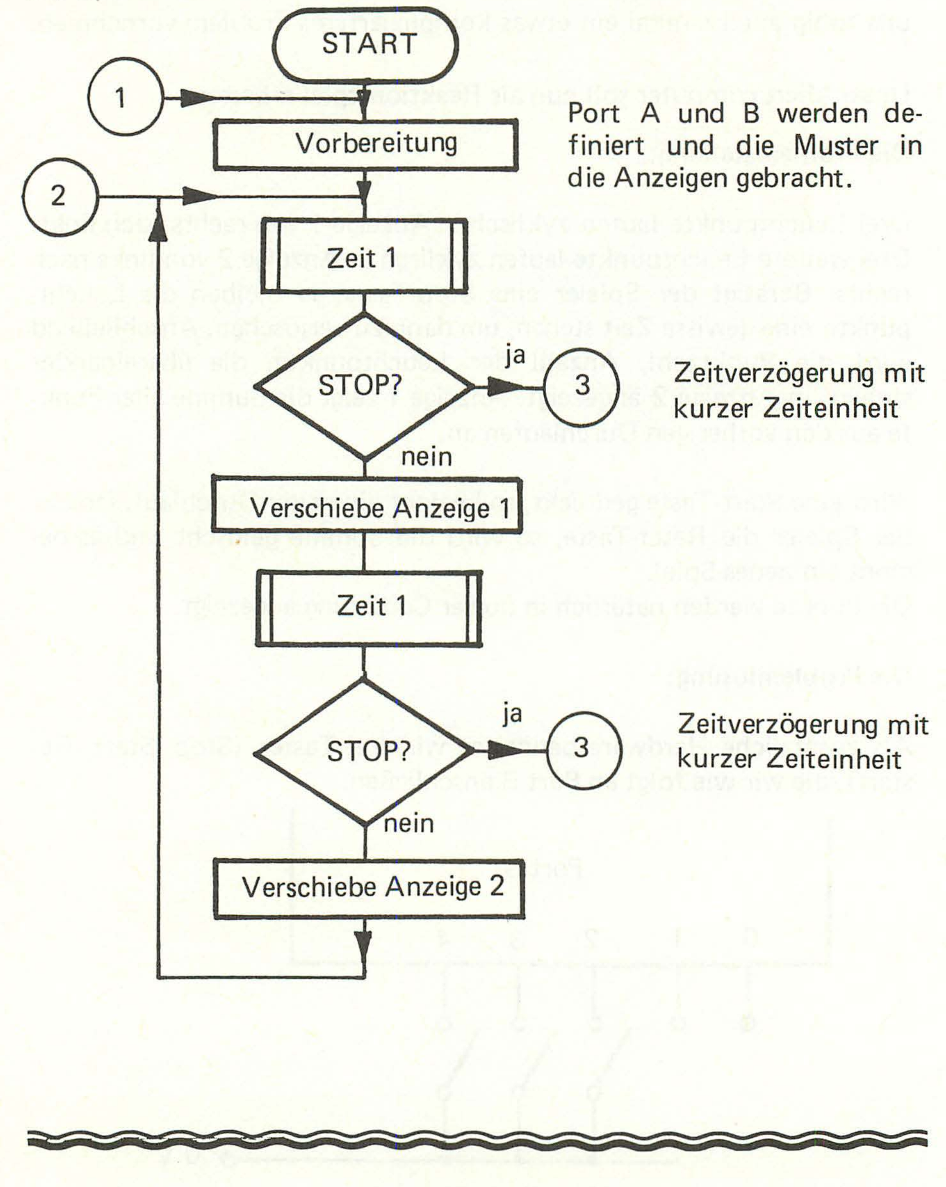
Wird eine Start-Taste gedrückt, so beginnt ein neuer Durchlauf. Drückt der Spieler die Reset-Taste, so wird die Summe gelöscht und es beginnt ein neues Spiel.
Die Punkte werden natürlich in dualer Codierung angezeigt.

Die Problemlösung:

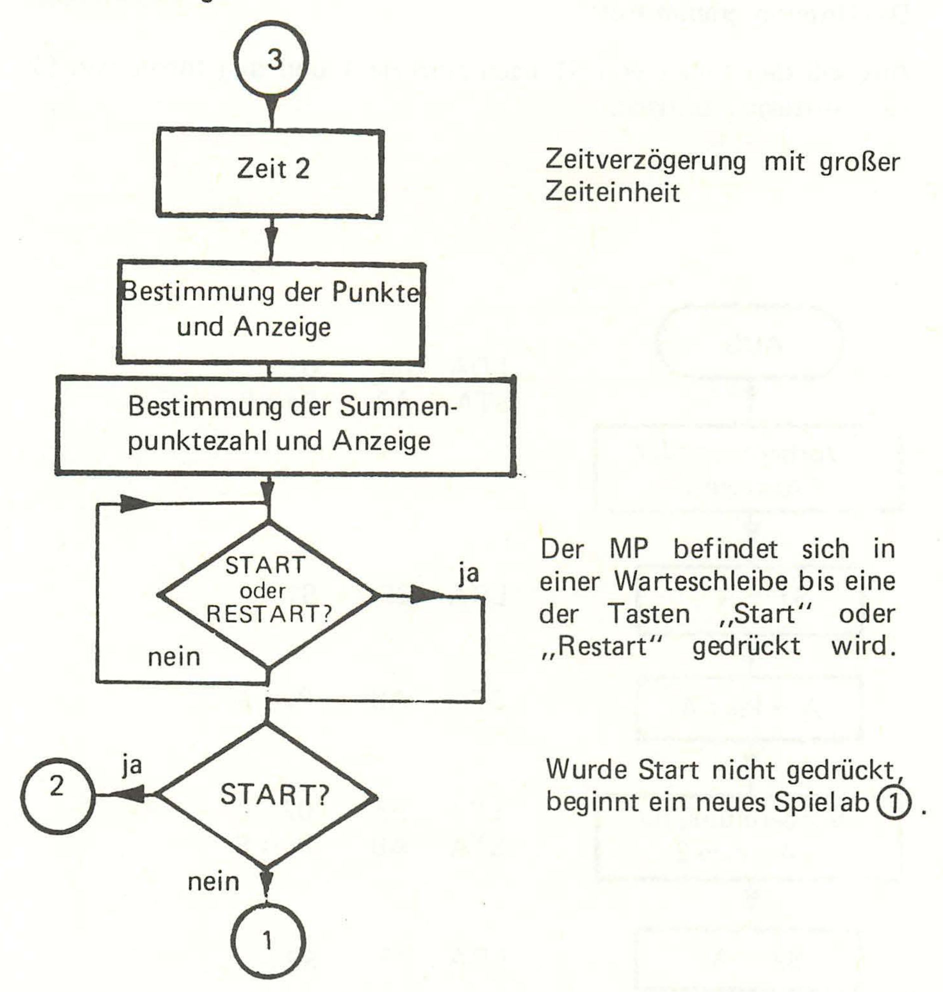
Als zusätzliche Hardware benötigen wir drei Tasten (Stop, Start, Restart), die wir wie folgt an Port B anschließen.



Der Aufbau des Programms soll an Hand eines groben Ablaufplanes erläutert werden.
Der Spielablauf:



Die Auswertung:

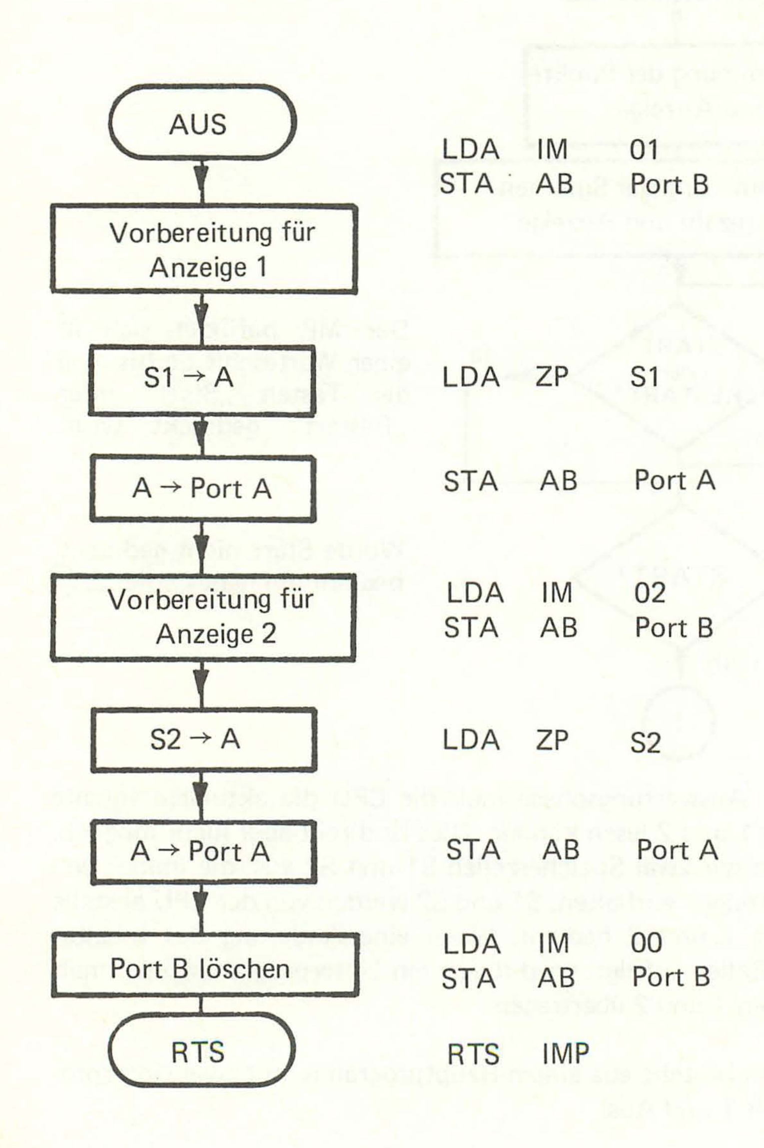


Während der Auswertungsphase muß die CPU die aktuellen Inhalte der Anzeigen 1 und 2 lesen können. Dies ist direkt aber nicht möglich. Daher wählen wir zwei Speicherzellen S1 und S2 aus, die immer den Inhalt der Anzeigen enthalten. S1 und S2 werden von der CPU anstelle der Anzeigen 1 und 2 bedient. Wenn eine Änderung des Inhaltes einer dieser Zellen erfolgt, wird durch ein Unterprogramm ihr Inhalt in die Anzeigen 1 und 2 übertragen.

Das Programm besteht aus einem Hauptprogramm mit zwei Unterprogrammen (Zeit 1 und Aus).

Das Unterprogramm Aus:

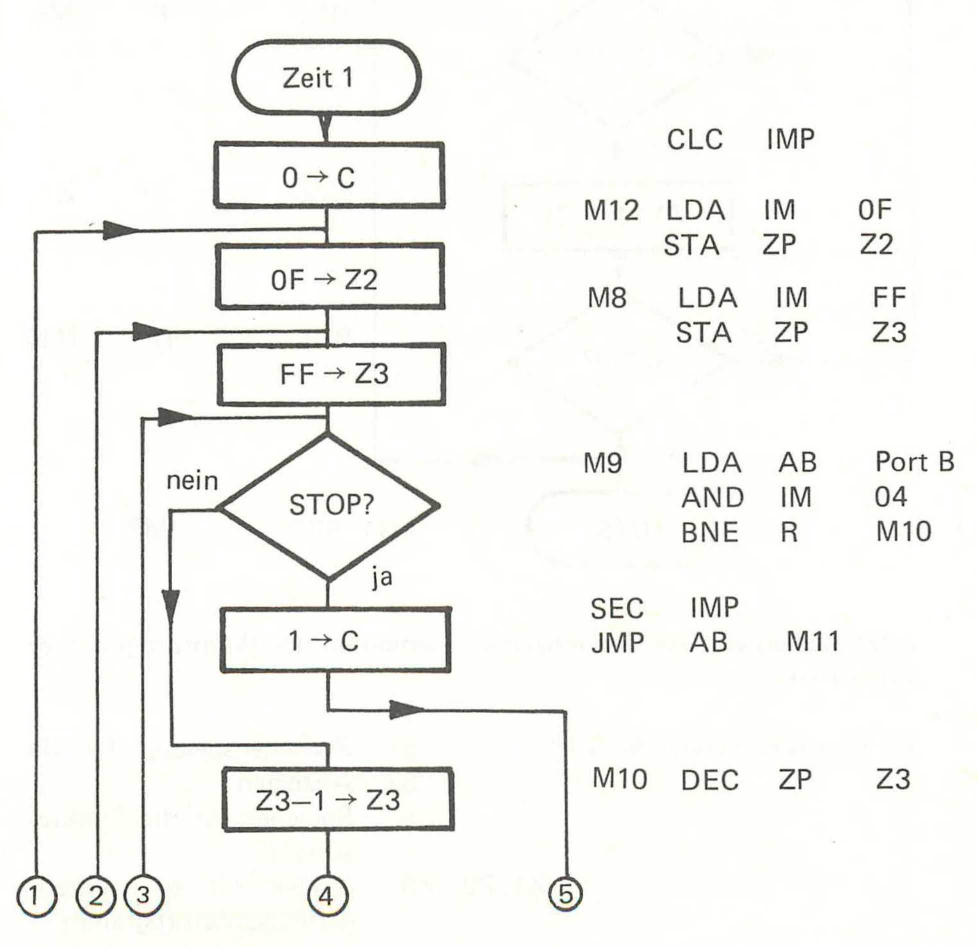
Aus soll den Inhalt von S1 nach Anzeige 1 und den Inhalt von S2 nach Anzeige 2 bringen.

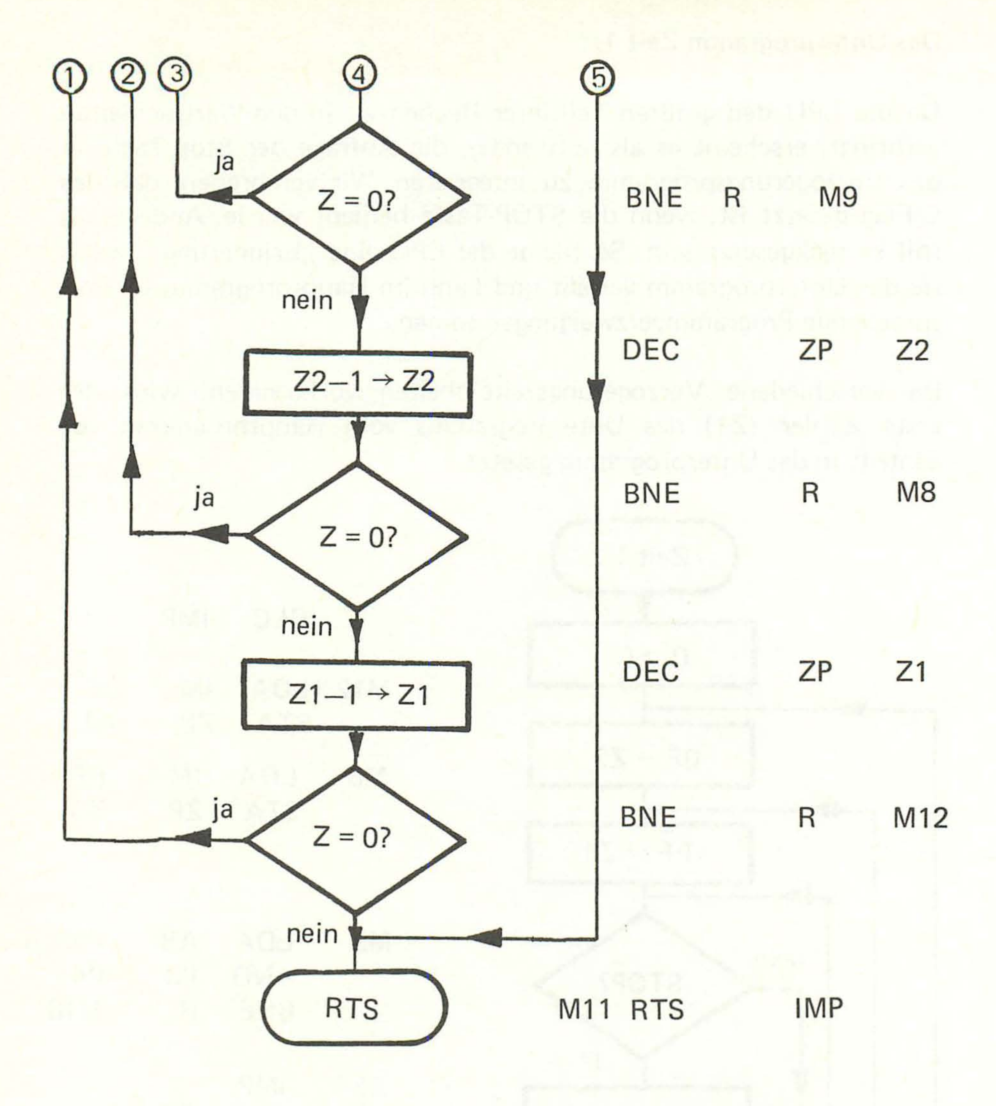


Das Unterprogramm Zeit 1:

Da die CPU den größten Teil ihrer Rechenzeit in den Warteschleifen verbringt, erscheint es als notwendig, die Abfrage der Stop-Taste in das Verzögerungsprogramm zu integrieren. Wir verabreden, daß das C-Flag gesetzt ist, wenn die STOP-Taste bedient wurde. Andernfalls soll es rückgesetzt sein. So bleibt der CPU eine „Erinnerung“, wenn sie das Unterprogramm verläßt und kann im Hauptprogramm für entsprechende Programmverzweigungen sorgen.

Da verschiedene Verzögerungszeiteinheiten vorkommen, wird der erste Zähler (Z1) des Unterprogramms vom Hauptprogramm vor Eintritt in das Unterprogramm gesetzt.





Jetzt wollen wir uns den einzelnen Elementen des Hauptprogrammes zuwenden.

An Variablen verwenden wir:

S1	Zwischenspeicher für die
S2	Anzeigen
S	Speicher für die Punktesumme
Z1, Z2, Z3	Zähler für das Verzögerungsunterprogramm

Die Vorbereitung:

```
      LDA  IM  FF
      STA  AB  PADD
      LDA  IM  03
      STA  AB  PBDD
  M7  LDA  IM  00
      STA  ZP  S
  M6  LDA  IM  07
      STA  ZP  S1
      LDA  IM  E0
      STA  ZP  S2
```

M6 und M7 sind Sprungziele für Programmsprünge aus anderen Programmteilen hierher.

Beide Rotationsbefehle des 6502 schieben die Information durch das C-Flag. Da die CPU immer zwischen Links- und Rechtsverschiebungen wechselt und im Unterprogramm Zeit 1 das C-Flag beeinflußt, muß dieses auf geeignete Weise zwischengespeichert werden. Vor jeder Rotation muß es wieder entsprechend gesetzt werden. Als Zwischenspeicher bietet sich der Kellerspeicher (Stack) an. Der Befehl PHP legt das Prozessorstatusregister, damit also auch das C-Flag, in den Stack ab.

Da zwei verschiedene Rotationen ausgeführt werden, müssen zwei Speicherzellen des Stacks reserviert werden. In der Vorbereitungsphase muß das C-Flag gelöscht und dann zweimal abgespeichert werden.

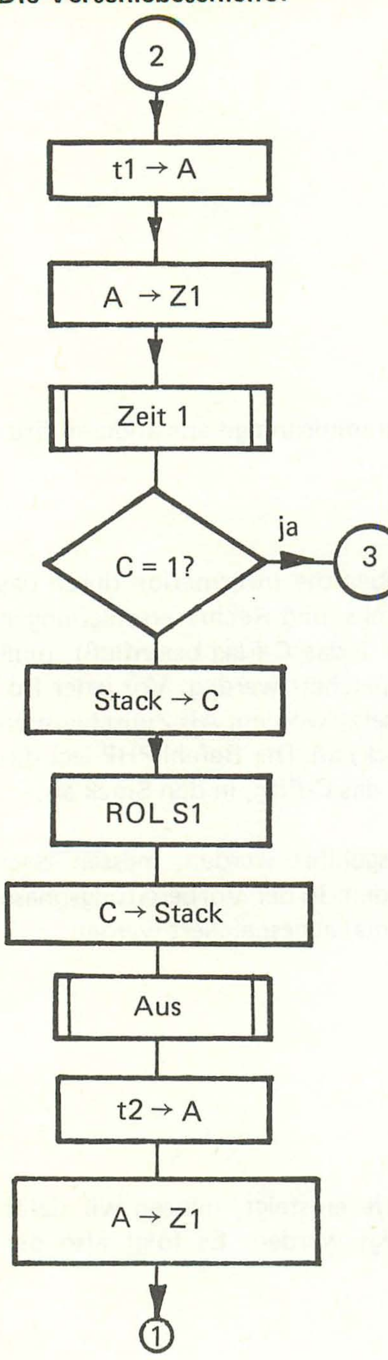
Es folgen die drei Befehle:

```
CLC  IMP
PHP  IMP
PHP  IMP
```

Bevor die CPU in die Verschiebeschleife einsteigt, müssen wir dafür Sorge tragen, daß S1 und S2 angezeigt werden. Es folgt also der Befehl:

```
JSR  AB  Aus
```

Die Verschiebeschleife:



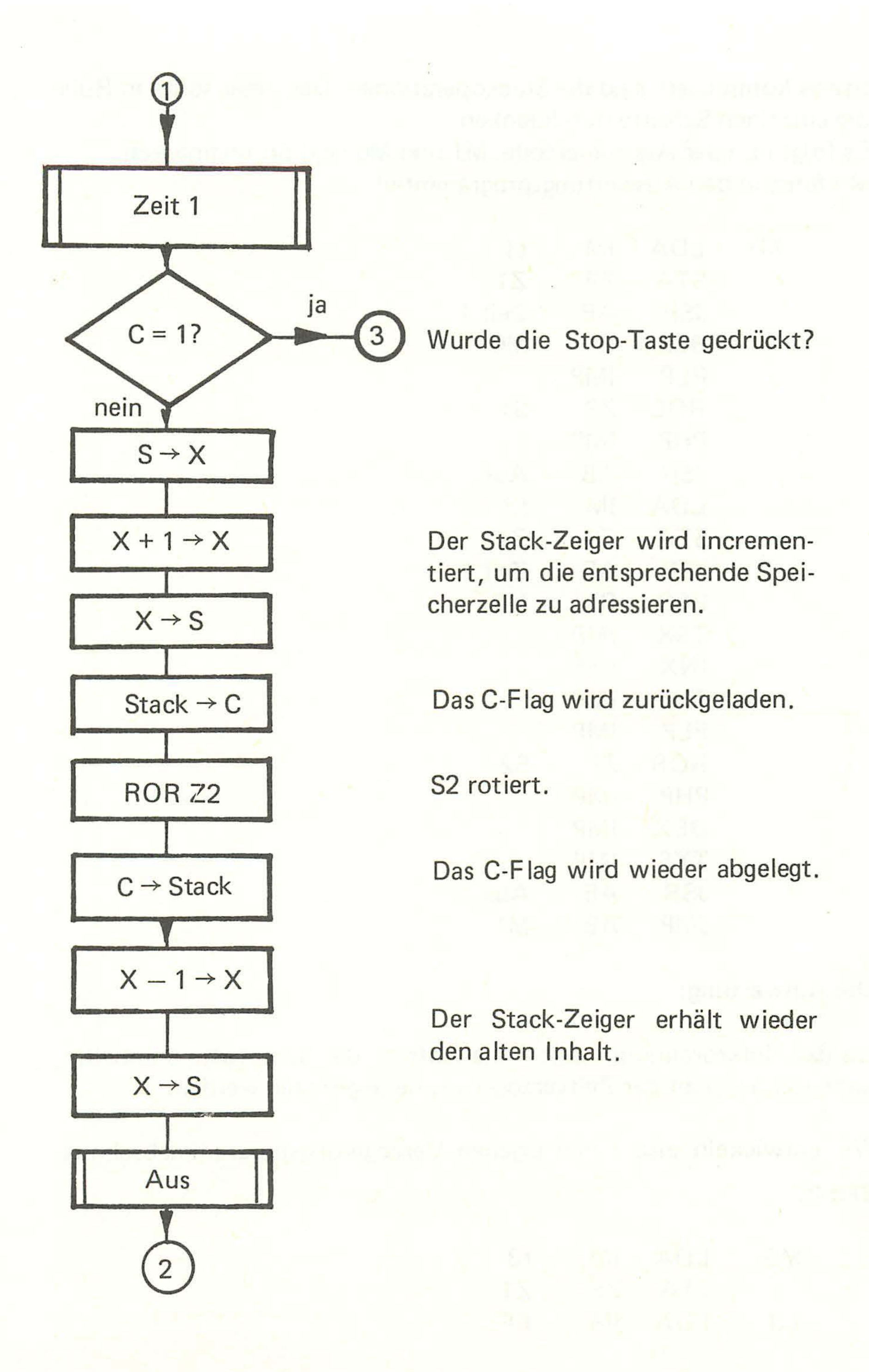
Die Zeitkonstante für die erste Zeitschleife wird gesetzt.

Wurde die Stop-Taste gedrückt? Wenn ja, springe in das Auswertungsprogramm.
Das C-Flag wird zurückgeladen.

S1 wird rotiert.

Das C-Flag wird zwischengespeichert.

Die Zeitkonstante für die zweite Zeitschleife wird gesetzt.


1
Zeit 1
C = 1?
ja
3
Wurde die Stop-Taste gedrückt?
nein
S → X
X + 1 → X
Der Stack-Zeiger wird incrementiert, um die entsprechende Speicherzelle zu adressieren.
X → S
Stack → C
Das C-Flag wird zurückgeladen.
ROR Z2
S2 rotiert.
C → Stack
Das C-Flag wird wieder abgelegt.
X – 1 → X
Der Stack-Zeiger erhält wieder den alten Inhalt.
X → S
Aus
2

Etwas kompliziert sind die Stackoperationen. Der Leser sollte in Ruhe die einzelnen Schritte durchdenken.
Es folgt nun der Assemblercode. M1 und M0 sind Sprungmarken. M0 führt in den Auswertungsprogrammteil.

```
M1  LDA  IM   t1
    STA  ZP   Z1
    JSR  AB   Zeit 1
    BCS  R    M0
    PLP  IMP
    ROL  ZP   S1
    PHP  IMP
    JSR  AB   Aus
    LDA  IM   t2
    STA  ZP   Z1
    JSR  AB   Zeit 1
    BCS  R    M0
    TSX  IMP
    INX  IMP
    TXS  IMP
    PLP  IMP
    ROR  ZP   S2
    PHP  IMP
    DEX  IMP
    TXS  IMP
    JSR  AB   Aus
    JMP  AB   M1
```

Die Auswertung:

Da das Unterprogramm Zeit 1 die Abfrage der Stop-Taste beinhaltet, kann es hier nicht zur Zeitverzögerung herangezogen werden.

Wir entwickeln also einen eigenen Verzögerungsprogrammabschnitt.

Zeit 2:

```
M0  LDA  IM   t3
    STA  ZP   Z1
L3  LDA  IM   FF
```

```
       STA  ZP   Z2
L2     LDA  IM   FF
       STA  ZP   Z3
L1     DEC  ZP   Z3
       BNE  R    L1
       DEC  ZP   Z2
       BNE  R    L2
       DEC  ZP   Z1
       BNE  R    L3
```

Die Bestimmung der Punktzahl:

S1 und S2 werden durch eine UND-Verknüpfung verbunden und dann die Anzahl der 1er ausgezählt. Das Y-Register wird als Zähler verwendet.

```
       CLC  IMP
       LDY  IM   00
       LDA  ZP   S1
       AND  ZP   S2
M3     BEQ  R    M2
       LSR  AC
       BCC  R    M3
       INY  IMP
       JMP  AB   M3
M2     CLC  IMP
       STY  ZP   S2
```

Die Bestimmung der Punktesumme:

Das ist ein einfaches Additionsprogramm. Das C-Flag wurde schon im vorherigen Abschnitt gelöscht. Es schließt sich das Unterprogramm Aus an.

```
TYA  IMP
ADC  ZP   S
STA  ZP   S1
STA  ZP   S
JSR  AB   Aus
```

Die Tastenabfrage und Verzweigung:

Die CPU muß jetzt warten, bis eine der Tasten „RESTART" oder „START" gedrückt wird. Sie bestimmt dann, welche Taste es war. Anschließend sorgt sie für die entsprechende Programmverzweigung.

M4	LDA	AB	Port B	Die Tasteninformation wird in A geladen und die R- bzw. S-Taste isoliert, ein Vergleich ausgeführt. Ein Rücksprung an den Schleifenanfang erfolgt, wenn keine Taste gedrückt wurde.
	AND	IM	18	
	CMP	IM	18	
	BEQ	R	M4	
	AND	IM	10	Die R-Taste wird isoliert.
	BNE	R	M5	
	JMP	AB	M7	Sprungziele in der Vorbereitung
M5	JMP	AB	M6	

Es wurden absolute Sprünge programmiert, da die Berechnung der relativen Adressen etwas umständlich ist.

Das Programm im Maschinencode (Hexcode):

Die Adressen:	S1 00 00	Z1 00 03
	S2 00 01	Z2 00 04
	S 00 02	Z3 00 05

Das Hauptprogramm:	02 00	bis	02 8B
Zeit 1:	02 8C	bis	02 AC
Aus:	02 AD	bis	02 C6

Das Programm im Maschinencode (Hexcode):

```
0200  a9  lda im  ff
0202  8d  sta abs 01  17
0205  a9  lda im  03
0207  8d  sta abs 03  17
020a  a9  lda im  00
```

```
020c  85  sta zp  02
020e  a9  lda im  07
0210  85  sta zp  00
0212  a9  lda im  e0
0214  85  sta zp  01
0216  18  clc imp
0217  08  php imp
0218  08  php imp
0219  20  jsr abs ad  02
021c  a9  lda im  01
021e  85  sta zp  03
0220  20  jsr abs 8c  02
0223  b0  bcs r   1f
0225  28  plp imp
0226  26  rol zp  00
0228  08  php imp
0229  20  jsr abs ad  02
022c  a9  lda im  01
022e  85  sta zp  03
0230  20  jsr abs 8c  02
0233  b0  bcs r   0f
0235  ba  tsx imp
0236  e8  inx imp
0237  9a  txs imp
0238  28  plp imp
0239  66  ror zp  01
023b  08  php imp
023c  ca  dex imp
023d  9a  txs imp
023e  20  jsr abs ad  02
0241  4c  jmp abs 1c  02
0244  a9  lda im  02
0246  85  sta zp  03
0248  a9  lda im  ff
024a  85  sta zp  04
024c  a9  lda im  ff
024e  85  sta zp  05
0250  c6  dec zp  05
0252  d0  bne r   fc
0254  c6  dec zp  04
0256  d0  bne r   f4
0258  c6  dec zp  03
025a  d0  bne r   ec
```

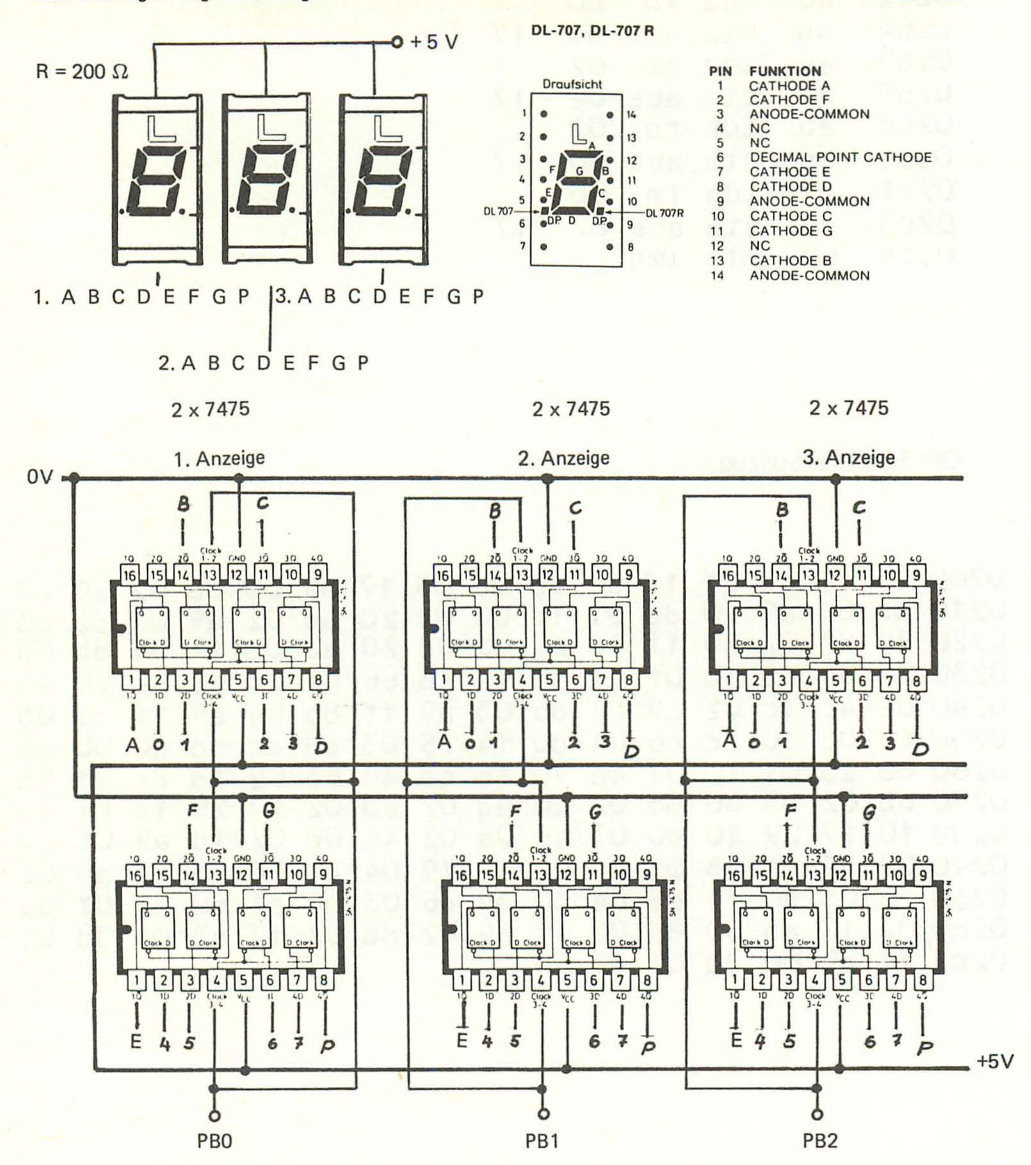
```
025c  18  clc imp
025d  a0  ldy im   00
025f  a5  lda zp   00
0261  25  and zp   01
0263  f0  beq r    07
0265  4a  lsr ac
0266  90  bcc r    fb
0268  c8  iny imp
0269  4c  jmp abs 63  02
026c  18  clc imp
026d  84  sty zp   01
026f  98  tya imp
0270  65  adc zp   02
0272  85  sta zp   00
0274  85  sta zp   02
0276  20  jsr abs ad  02
0279  ad  lda abs 02  17
027c  29  and im   18
027e  c9  cmp im   18
0280  f0  beq r    f7
0282  29  and im   10
0284  d0  bne r    03
0286  4c  jmp abs 0a  02
0289  4c  jmp abs 0e  02
028c  18  clc imp
028d  a9  lda im   0f
028f  85  sta zp   04
0291  a9  lda im   ff
0293  85  sta zp   05
0295  ad  lda abs 02  17
0298  29  and im   04
029a  d0  bne r    04
029c  38  sec imp
029d  4c  jmp abs ad  02
02a0  c6  dec zp   05
02a2  d0  bne r    f1
02a4  c6  dec zp   04
02a6  d0  bne r    e9
02a8  c6  dec zp   03
02aa  d0  bne r    e1
02ac  60  rts imp
02ad  a9  lda im   01
02af  8d  sta abs 02  17
```

```
02b2   a5   lda zp  00
02b4   8d   sta abs 00  17
02b7   a9   lda im  02
02b9   8d   sta abs 02  17
02bc   a5   lda zp  01
02be   8d   sta abs 00  17
02c1   a9   lda im  00
02c3   8d   sta abs 02  17
02c6   60   rts imp
```

Der Speicherauszug:

```
0200 a9 ff 8d 01 17 a9 03 8d 03 17 a9 00 85 02 a9 07
0210 85 00 a9 e0 85 01 18 08 08 20 ad 02 a9 01 85 03
0220 20 8c 02 b0 1f 28 26 00 08 20 ad 02 a9 01 85 03
0230 20 8c 02 b0 0f ba e8 9a 28 66 01 08 ca 9a 20 ad
0240 02 4c 1c 02 a9 02 85 03 a9 ff 85 04 a9 ff 85 05
0250 c6 05 d0 fc c6 04 d0 f4 c6 03 d0 ec 18 a0 00 a5
0260 00 25 01 f0 07 4a 90 fb c8 4c 63 02 18 84 01 98
0270 65 02 85 00 85 02 20 ad 02 ad 02 17 29 18 c9 18
0280 f0 f7 29 10 d0 03 4c 0a 02 4c 0e 02 18 a9 0f 85
0290 04 a9 ff 85 05 ad 02 17 29 04 d0 04 38 4c ad 02
02a0 c6 05 d0 f1 c6 04 d0 e9 c6 03 d0 e1 60 a9 01 8d
02b0 02 17 a5 00 8d 00 17 a9 02 8d 02 17 a5 01 8d 00
02c0 17 a9 00 8d 02 17 60
```

Eine 3-stellige 7-Segmentanzeige **Abb. 4.6.1**



6 x 7475

Die Portanschlüsse 0–7 vom Port A werden mit den Anschlüssen 1–7 an den IC's verbunden.

In die Anschlußleisten für die 7-Segment-Anzeige a b c d e f g p müssen pro Anzeige je 8 Widerstände 200 Ohm/0,25 W geschaltet werden.

4.6 Programme für eine dreistellige Siebensegmentanzeige

Eine weitere Zusatzhardware für unseren Microcomputer ist eine Siebensegmentanzeige. Wir wollen gleich eine dreistellige Anzeige entwickeln, damit wir auch umfangreichere Programme entwerfen können.

Der Leser, der dieses Buch bisher gut durchgearbeitet hat, wird sicherlich einfache Programme zu dieser Anzeige selber, ohne weitere Hilfestellungen, entwickeln können. Daher soll nur im ersten Beispiel eine einfache Anwendung vorgestellt werden. Alle folgenden Programmbeispiele wurden bewußt etwas anspruchsvoll gestaltet. Der Leser soll ein Gefühl dafür entwickeln, welcher Softwareaufwand für welche Problemstellungen notwendig ist.

Die Hardware ist wieder denkbar einfach. Die drei Siebensegmentanzeigen werden an je einen Zwischenspeicher angeschlossen. Diese Speicher bestehen aus je zwei 7475 Bausteinen. Die Eingänge sind parallel geschaltet und an den Ausgangsport A gelegt. Die Takteingänge werdenan den zweiten Ausgangsport(Port B) angeschlossen; der Takteingang von Speicher 1 an Bit 0, der von Speicher 2 an Bit 1 und der letztere an Bit 2.

In der Abbildung 4.6.1 ist die Schaltung für Anzeigen mit gemeinsamer Anode abgebildet. Die Anzeigen 1 bis 3 wurden von links nach rechts angeordnet. Anzeige 3 hat also die niedrigste Wertigkeit.

1. Die Hexdarstellung von Daten

In diesem ersten einfachen Beispiel soll das Unterprogramm zur Bedienung der Anzeige entwickelt werden. Dieses Unterprogramm ist etwas anspruchsvoller als es zur Erfüllung der hier gestellten Aufgabe notwendig wäre. Wir wollen es aber unverändert in den folgenden Beispielen verwenden und entscheiden uns daher für eine komfortablere Version.

Das Unterprogramm zur Bedienung der Anzeige

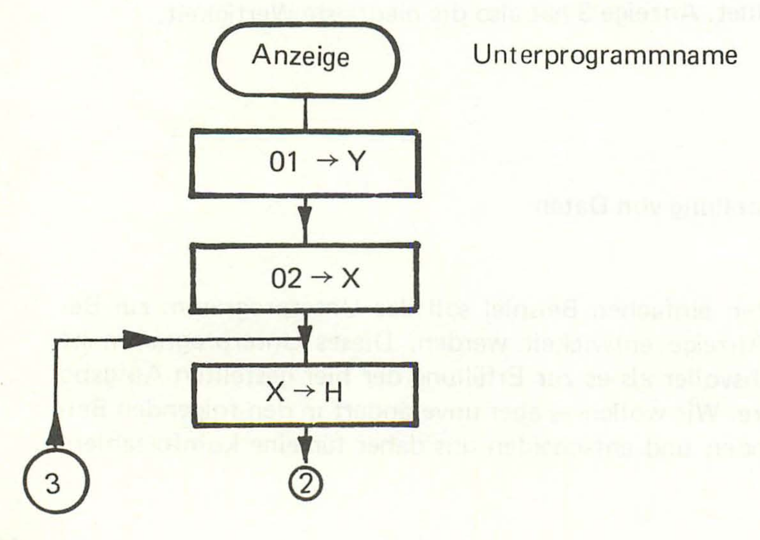
Die Aufgabenstellung:
In den Speicherzellen Z1, Z2 und Z3, die die Adressen 00 00 bis 00 02 haben, befinden sich die Hexzahlen, die in die Anzeigen 3 bis 1 übertragen werden. Bei der Übertragung soll ein Überschreiben (Zwei Anzeigen werden gleichzeitig geladen) vermieden werden.

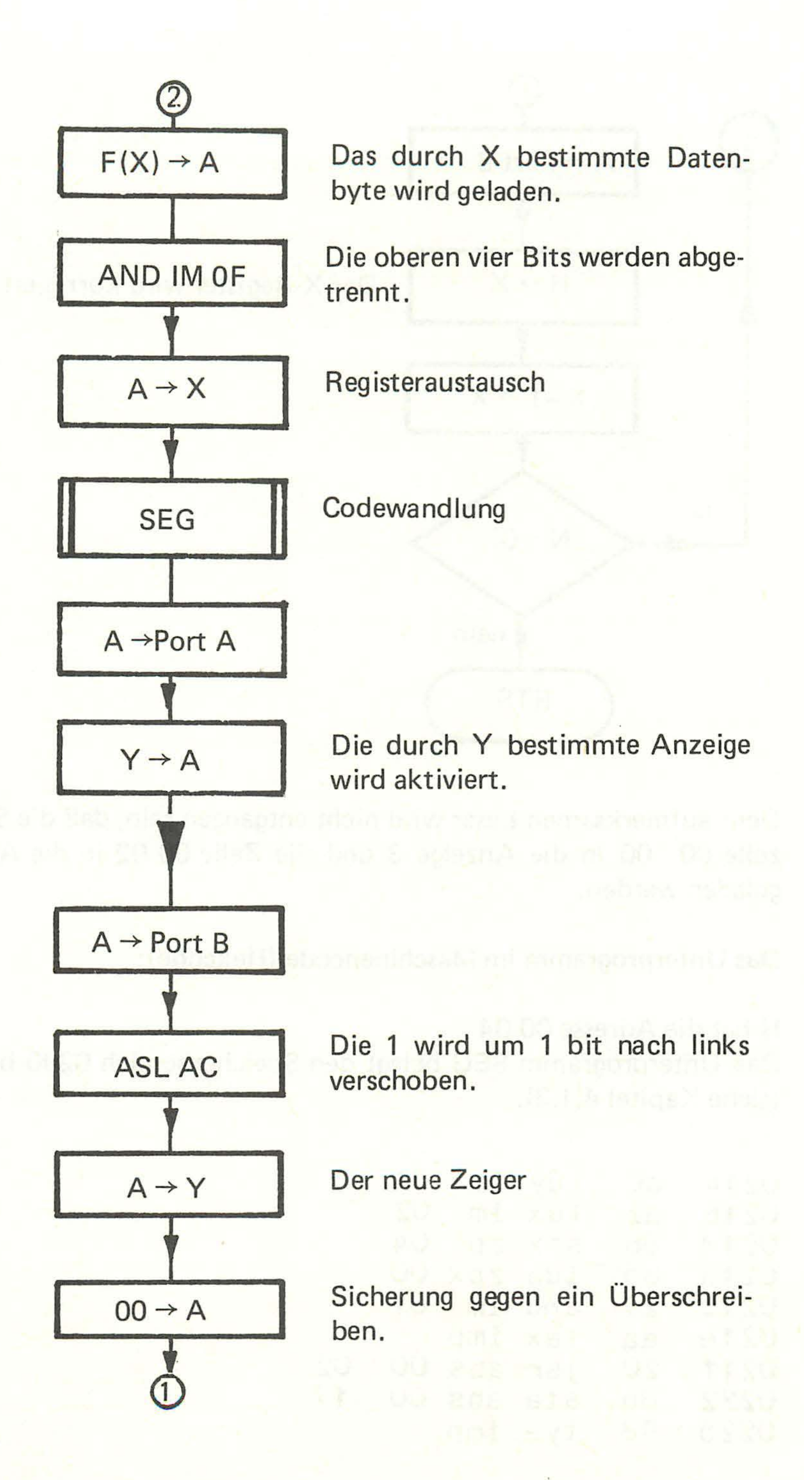
Die Problemanalyse:

Bevor ein Datenbyte in eine Anzeige geschrieben wird, muß eine Codewandlung vom Hex- in den 7-Segmentcode durchgeführt werden. Wir verwenden dazu das in Kapitel 4.1.3 besprochene Programm SEG.

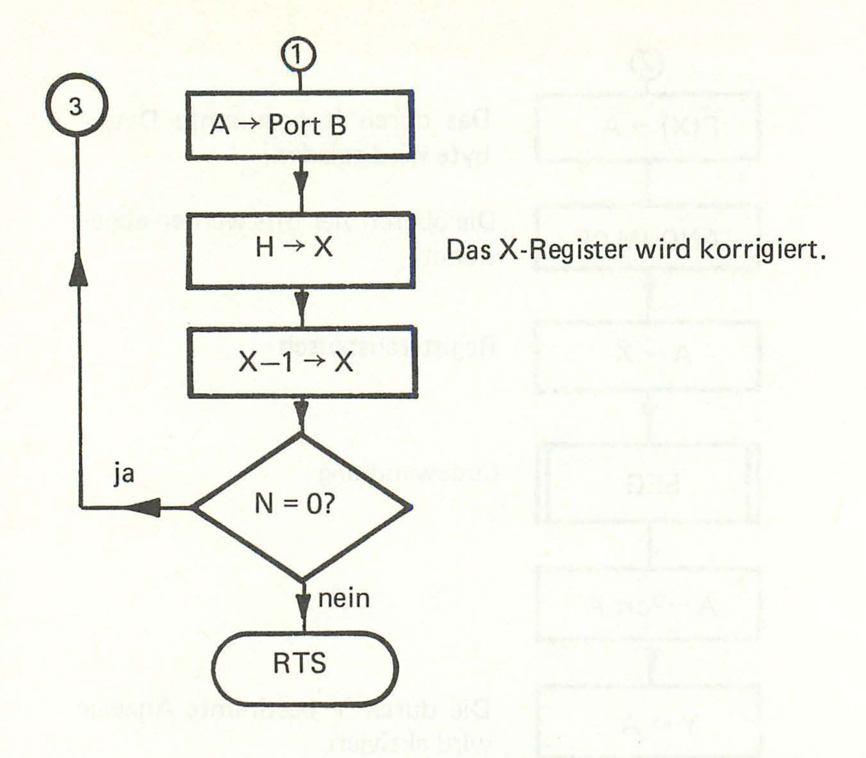
Der Übersetzungs- und Einschreibevorgang wiederholt sich dreimal mit nur leichten Abänderungen. So erscheint es sinnvoll, das Programm als eine Schleife mit drei Durchläufe aufzubauen. Die Datenbytes Z1 bis Z3 bilden ein Datenfeld. Das X-Register wird als Zähler und Datenzeiger eingesetzt. Das Y-Register beinhaltet die Information, die die entsprechende Anzeige aktiviert.

Da das Programm SEG das X-Register verändert, muß noch ein Hilfsspeicher H festgelegt werden, der den Inhalt von X kurzzeitig aufnimmt.





2
F(X) → A
Das durch X bestimmte Datenbyte wird geladen.
AND IM 0F
Die oberen vier Bits werden abgetrennt.
A → X
Registeraustausch
SEG
Codewandlung
A →Port A
Y → A
Die durch Y bestimmte Anzeige wird aktiviert.
A → Port B
ASL AC
Die 1 wird um 1 bit nach links verschoben.
A → Y
Der neue Zeiger
00 → A
Sicherung gegen ein Überschreiben.
1



Dem aufmerksamen Leser wird nicht entgangen sein, daß die Speicherzelle 00 00 in die Anzeige 3 und die Zelle 00 02 in die Anzeige 1 geladen werden.

Das Unterprogramm im Maschinencode (Hexcode):

H hat die Adresse 00 04
Das Unterprogramm SEG belegt den Speicherbereich 0200 bis 02 13 (siehe Kapitel 4.1.3).

```
0214   a0   ldy im  01
0216   a2   ldx im  02
0218   86   stx zp  04
021a   b5   lda zpx 00
021c   29   and im  0f
021e   aa   tax imp
021f   20   jsr abs 00  02
0222   8d   sta abs 00  17
0225   98   tya imp
```

```
0226   8d   sta abs 02   17
0229   0a   asl ac
022a   a8   tay imp
022b   a9   lda im  00
022d   8d   sta abs 02   17
0230   a6   ldx zp  04
0232   ca   dex imp
0233   10   bpl r   e3
0235   60   rts imp
```

Der Speicherauszug:

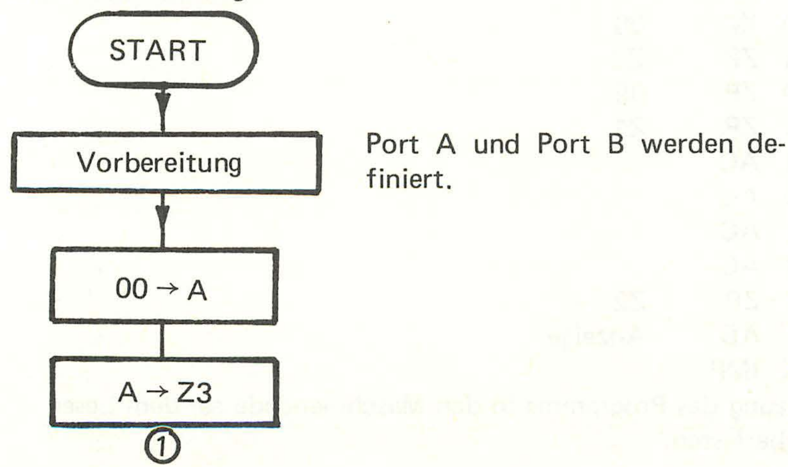
```
0214 a0 01 a2 02 86 04 b5 00 29 0f aa 20 00 02 8d 00
0224 17 98 8d 02 17 0a a8 a9 00 8d 02 17 a6 04 ca 10
0234 e3 60
```

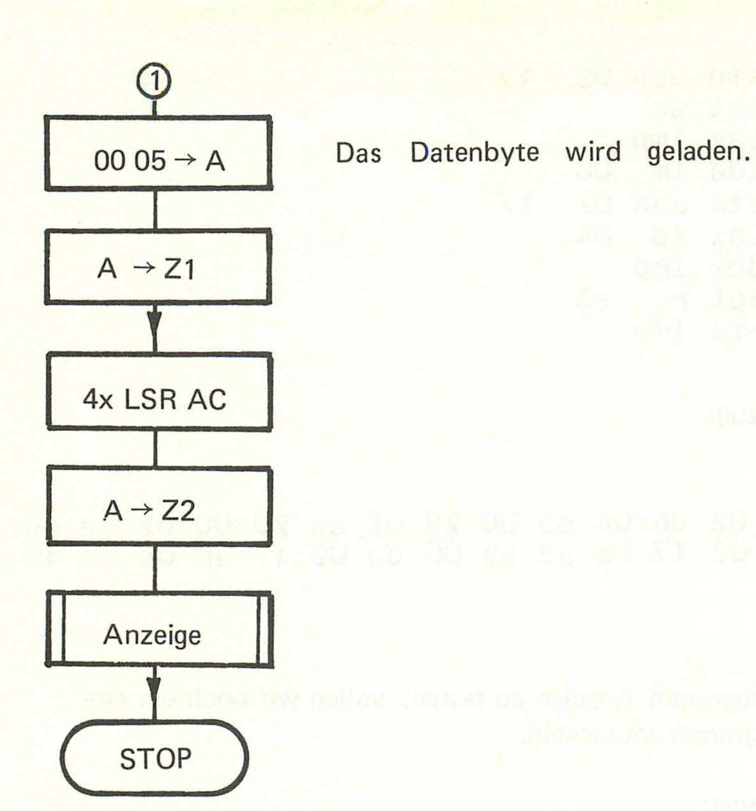
Um das Unterprogramm Anzeige zu testen, wollen wir noch ein einfaches Hauptprogramm entwickeln.

Die Problemstellung:

Das Datenbyte mit der Adresse 00 05 soll in Anzeige 3 und Anzeige 2 als Hexzeichen angezeigt werden. Anzeige 1 soll eine Null beinhalten.

Die Problemlösung:





Das Programm im Assemblercode:

```
LDA  IM   FF
STA  AB   PADD
STA  AB   PBDD
LDA  IM   00
STA  ZP   Z3
LDA  ZP   05
STA  ZP   Z1
LSR  AC
LSR  AC
LSR  AC
LSR  AC
STA  ZP   Z2
JSR  AB   Anzeige
BRK  IMP
```

Die Übersetzung des Programms in den Maschinencode sei dem Leser zur Übung überlassen.

2. Ein Zählautomat

Sehr häufig wird an Steuerungsanlagen die Aufgabe gestellt, eine Zählung durchzuführen.

Natürlich ist ein einfacher Zählautomat mit billigen TTL-Bausteinen leicht aufgebaut. Soll der Zähler aber mehrere Vergleiche mit vorgegebenen Zählzuständen durchführen und dann bestimmte Operationen einleiten und überwachen, so ist eine entsprechende TTL-Schaltung schon recht aufwendig.
Ein Microcomputer kann mit geeigneter Hardware entsprechende Aufgabenstellungen wesentlich besser und einfacher erfüllen.

Die Problemstellung:

Der Microcomputer soll Impulse, die an dem Impulseingang Z anliegen, auszählen. Falls diese Impulse nicht entprellt sind, muß eine Softwareentprellung vorgesehen werden. Eine Programmerweiterung für zusätzliche Operationen, die vom Zählerzustand beeinflußt werden, sollte möglich sein.

Die Problemanalyse:

Den Impulseingang kann man an Bit 3 des Port B legen, wenn dieser bidirektional und getrennt definierbar ist (wie beim KIM).

Wir wollen die Impulse mit einem Taster, den wir an Bit 3 des Ports B anschließen, erzeugen.

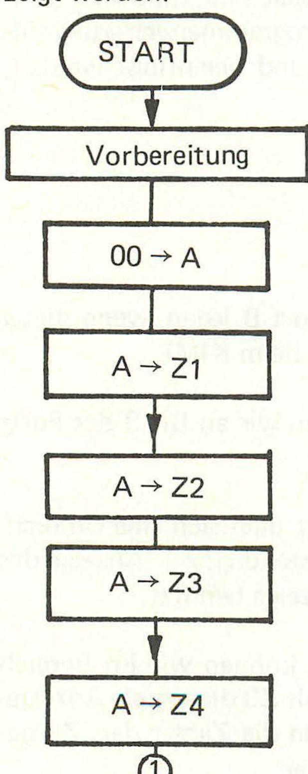
Gezählt wird im Dezimalmode. Dadurch spart man sich eine Umrechnung von der Dualdarstellung in die BCD-Darstellung. Zur Anzeige des Zählerzustandes wird das Unterprogramm Anzeige benutzt.

Da wir eine dreistellige Anzeige verwenden, können wir im Bereich von 0 bis 999 zählen. Die Speicherzellen Z1 bis Z3 dienen als Anzeigeregister. Die Speicherzellen Z4 und Z5 stellen die Zähler dar. Z4 beinhaltet die Einer und Zehner, Z5 die Hunderter.

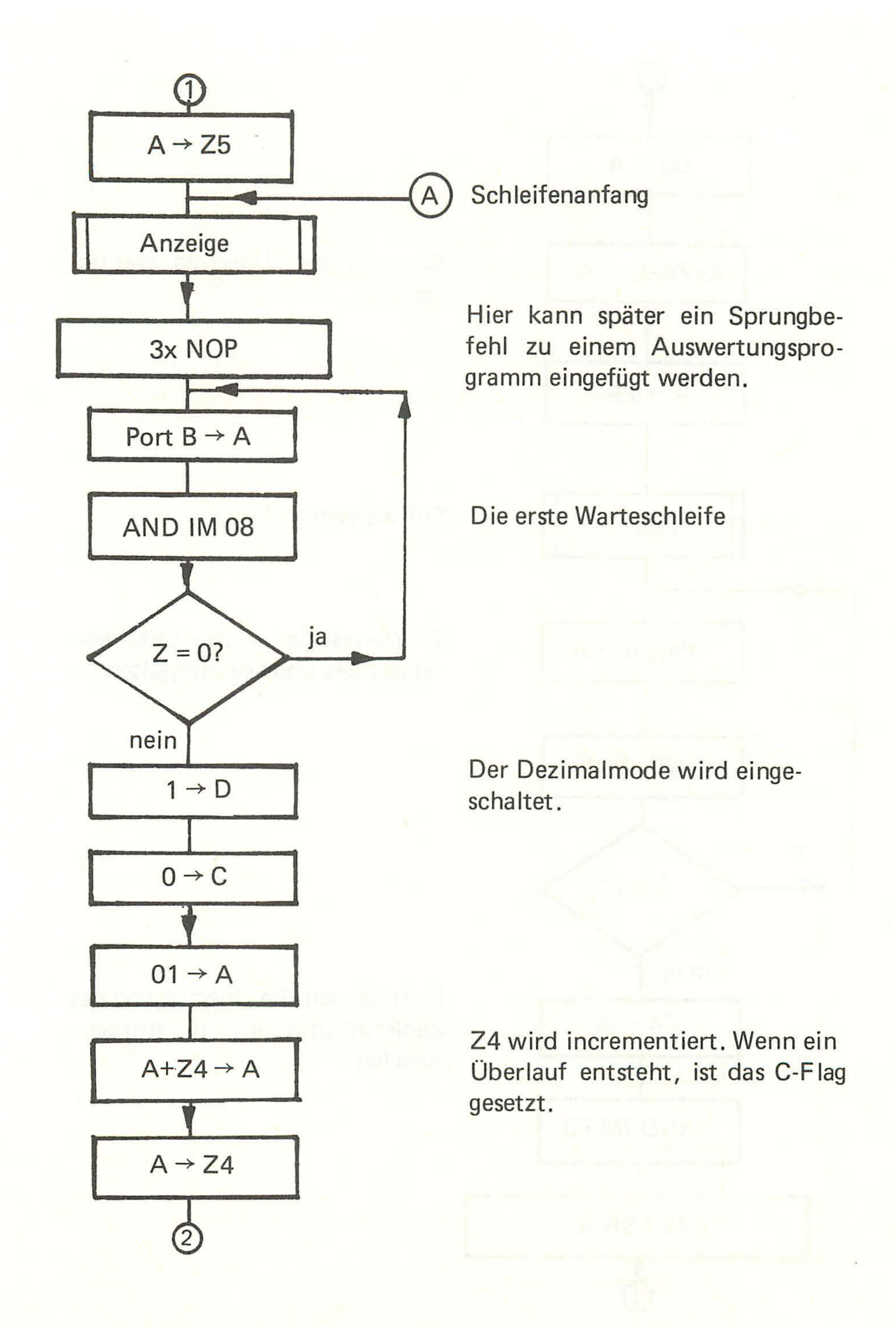
Das Zählprogramm ist eine Endlosschleife. Pro Durchlauf der Schleife wird der Zähler incrementiert. Bevor die CPU einen Durchlauf beginnt, befindet sie sich in einem Wartezustand. Sie wartet auf einen Impuls.

Nachdem sie gezählt hat, muß sie wieder warten. Jetzt muß das Ende des entsprechenden Impulses bestimmt werden. Wenn die Impulslänge groß gegenüber der Rechenzeit der CPU ist, würde sie sonst mehrere Durchläufe pro Impuls ausführen.
Sind die Impulse nicht prellfrei, muß der Programmierer vor dieser zweiten Warteschleife für eine geeignete Softwareentprellung sorgen.

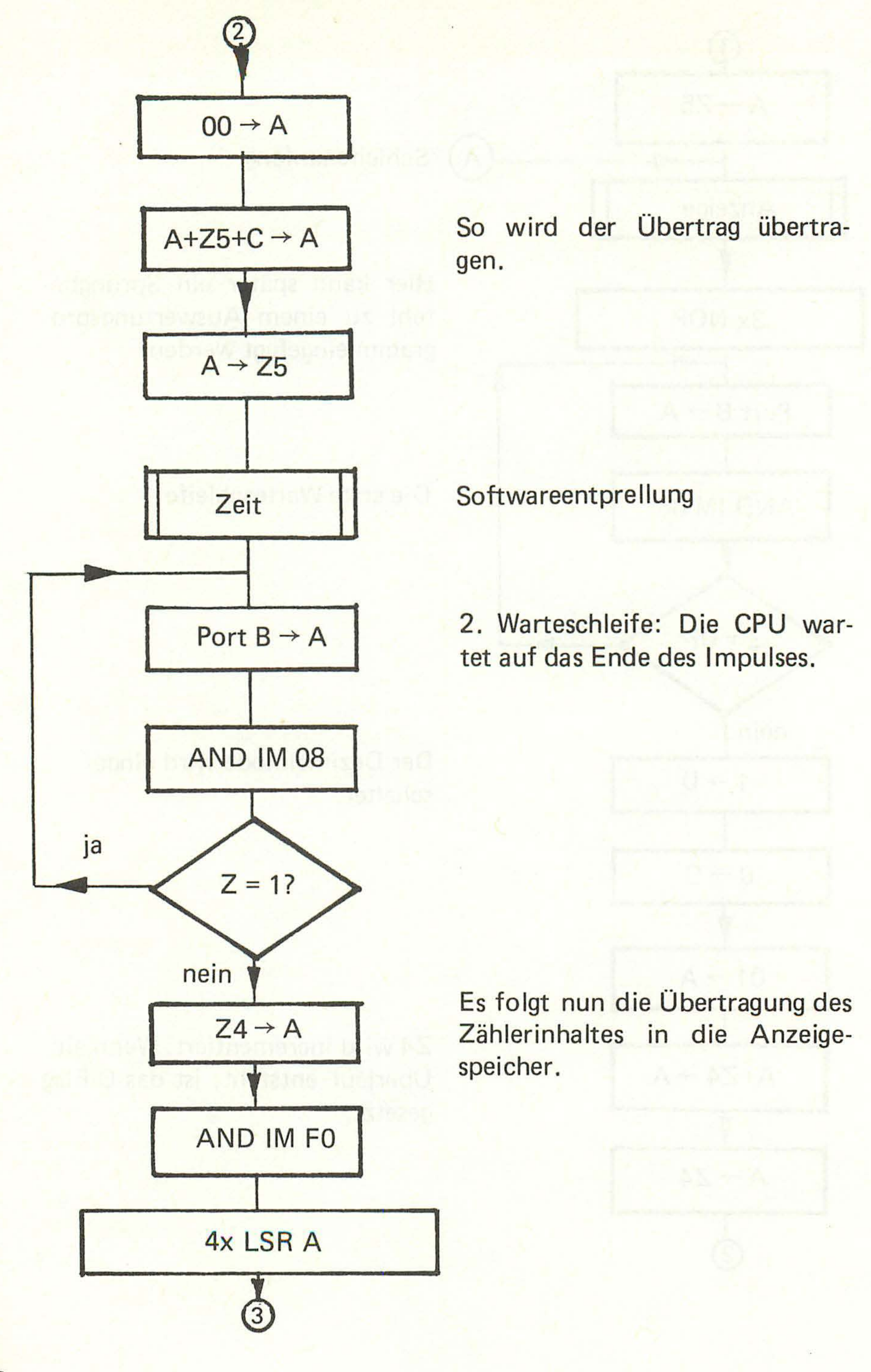
Da wir mit einem einfachen Taster die Impulse erzeugen, müssen wir eine Entprellung vorsehen. Dafür sorgt ein Verzögerungsunterprogramm Zeit.
Natürlich soll bei jedem Zählschleifendurchgang der Zählerstand angezeigt werden.



Die Anzeigenzellen und Zähler werden gelöscht.



1
A → Z5
A
Schleifenanfang
Anzeige
3x NOP
Hier kann später ein Sprungbefehl zu einem Auswertungsprogramm eingefügt werden.
Port B → A
AND IM 08
Die erste Warteschleife
Z = 0?
ja
nein
1 → D
Der Dezimalmode wird eingeschaltet.
0 → C
01 → A
A+Z4 → A
Z4 wird incrementiert. Wenn ein Überlauf entsteht, ist das C-Flag gesetzt.
A → Z4
2

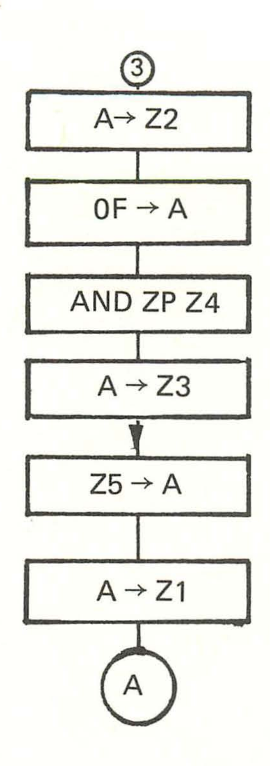


So wird der Übertrag übertragen.

Softwareentprellung

2. Warteschleife: Die CPU wartet auf das Ende des Impulses.

Es folgt nun die Übertragung des Zählerinhaltes in die Anzeigespeicher.



Zum Schleifenanfang

Das Unterprogramm Zeit ist leicht aufgebaut. Die Register X und Y werden als Schleifenzähler in einer Zweifachschleife benutzt.

Das Programm im Maschinencode (Hexcode):

Die Adressen:	Z3	00 00	Z4	00 03
	Z2	00 01	Z5	00 05
	Z1	00 02	H	00 04

Anzeige: 02 14

Zeit: 02 88

① 02 4C

Das Programm im Maschinencode (Hexcode):

```
0236   a9   lda im  ff
0238   8d   sta abs 01  17
023b   a9   lda im  07
```

```
023d  8d  sta abs 03  17
0240  a9  lda im  00
0242  85  sta zp  00
0244  85  sta zp  01
0246  85  sta zp  02
0248  85  sta zp  03
024a  85  sta zp  05
024c  20  jsr abs 14  02
024f  ea  nop imp
0250  ea  nop imp
0251  ea  nop imp
0252  ad  lda abs 02  17
0255  29  and im  08
0257  d0  bne r   f9
0259  f8  sed imp
025a  18  clc imp
025b  a9  lda im  01
025d  65  adc zp  03
025f  85  sta zp  03
0261  a9  lda im  00
0263  65  adc zp  05
0265  85  sta zp  05
0267  20  jsr abs 88  02
026a  ad  lda abs 02  17
026d  29  and im  08
026f  f0  beq r   f9
0271  a5  lda zp  03
0273  29  and im  f0
0275  4a  lsr ac
0276  4a  lsr ac
0277  4a  lsr ac
0278  4a  lsr ac
0279  85  sta zp  01
027b  a9  lda im  0f
027d  25  and zp  03
027f  85  sta zp  00
0281  a5  lda zp  05
0283  85  sta zp  02
0285  4c  jmp abs 4c  02
0288  a0  ldy im  01
028a  a2  ldx im  0f
028c  ca  dex imp
028d  d0  bne r   fd
```

```
028f   88   dey imp
0290   d0   bne r    f8
0292   60   rts imp
```

Der Speicherauszug:

```
0236 a9 ff 8d 01 17 a9 07 8d 03 17 a9 00 85 00 85 01
0246 85 02 85 03 85 05 20 14 02 ea ea ea ad 02 17 29
0256 08 d0 f9 f8 18 a9 01 65 03 85 03 a9 00 65 05 85
0266 05 20 88 02 ad 02 17 29 08 f0 f9 a5 03 29 f0 4a
0276 4a 4a 4a 85 01 a9 0f 25 03 85 00 a5 05 85 02 4c
0286 4c 02 a0 01 a2 0f ca d0 fd 88 d0 f8 60
```

3. Ein Reaktionstester (Stopuhr)

Das vorherige Programmbeispiel muß nur wenig abgeändert werden, um eine Stopuhr oder einen Reaktionstester zu realisieren.

Die Problemstellung:

Unsere Stop-Uhr soll drei Tasten haben. Eine Start-, eine Stop- und eine Reset-Taste. Die Anzeige der Uhr soll nicht mitlaufen, sondern nur nach Betätigung der Stop-Taste die Zeit anzeigen.

Die Problemanalyse:

Wir schließen die Tasten an den Port B an, da dieser bidirektional ist; die Starttaste an Bit 3, die Stopptaste an Bit 4 und die Resettaste an Bit 5.

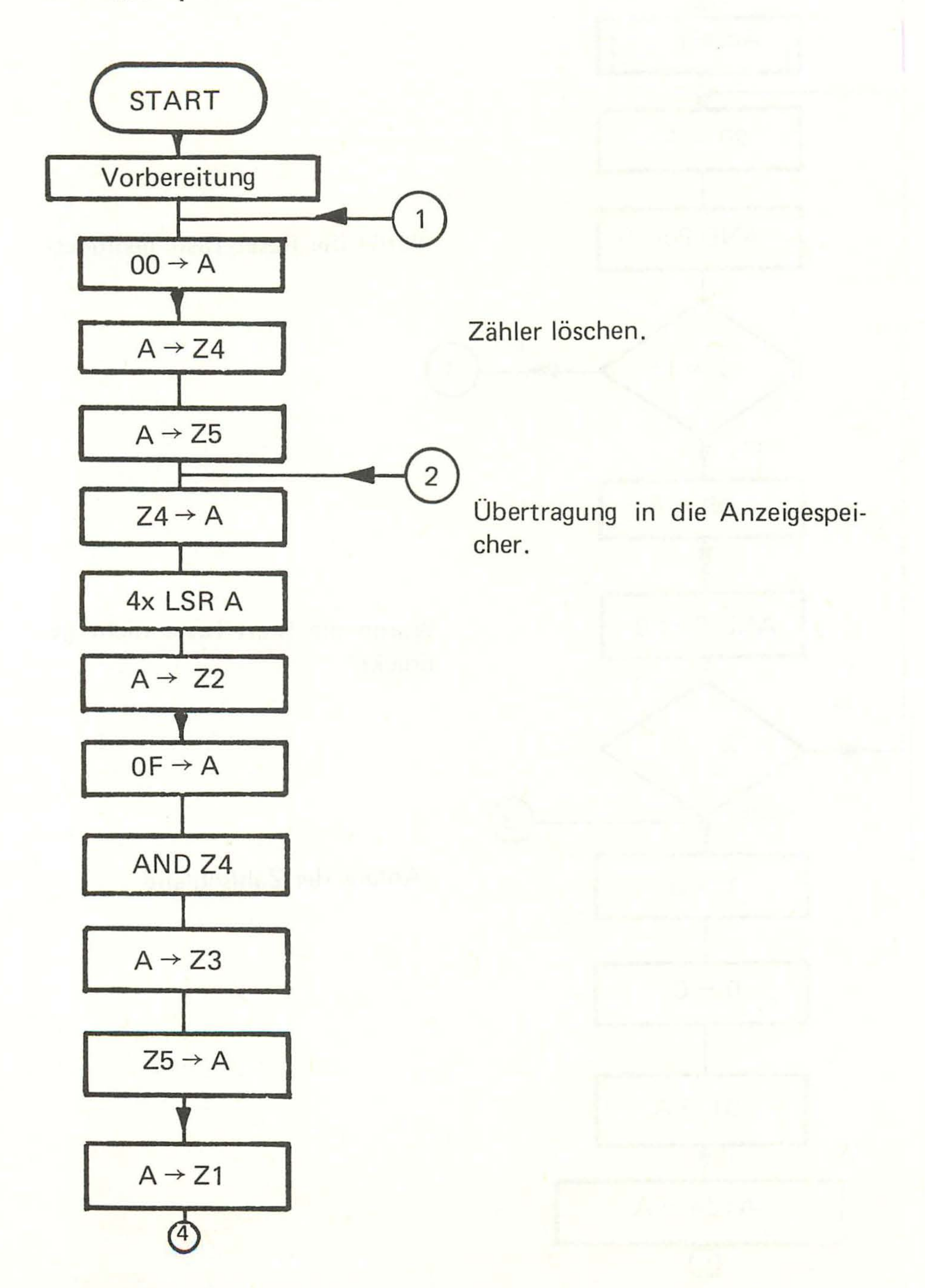
Da wir das Unterprogramm Anzeige verwenden, werden die Speicherzellen 0000 bis 0002 durch Z3 bis Z1 belegt.Das Programm besteht im wesentlichen wieder aus einer Zählschleife. Als Zähler benutzen wir wieder die Zellen Z4 und Z5. Gezählt wird im Dezimalmode.

Die benötigte Zeit für einen Schleifendurchgang wird durch das Verzögerungsprogramm Zeit bestimmt. Wählt man in Zeit die Konstanten geeignet, so kann man zum Beispiel eine Stopuhr mit der Auflösung von 1/10 Sekunden programmieren. Man kann natürlich sehr viel größere und kleinere Zeiteinheiten wählen.

Bei jedem Schleifendurchgang muß die Stop-Taste abgefragt werden. Wird sie betätigt, so soll die CPU die Zählschleife verlassen und den Zählerstand anzeigen. Anschließend begibt sie sich in eine Warteschleife. Sie wartet ab, ob die Reset- oder die Start-Taste betätigt wird.

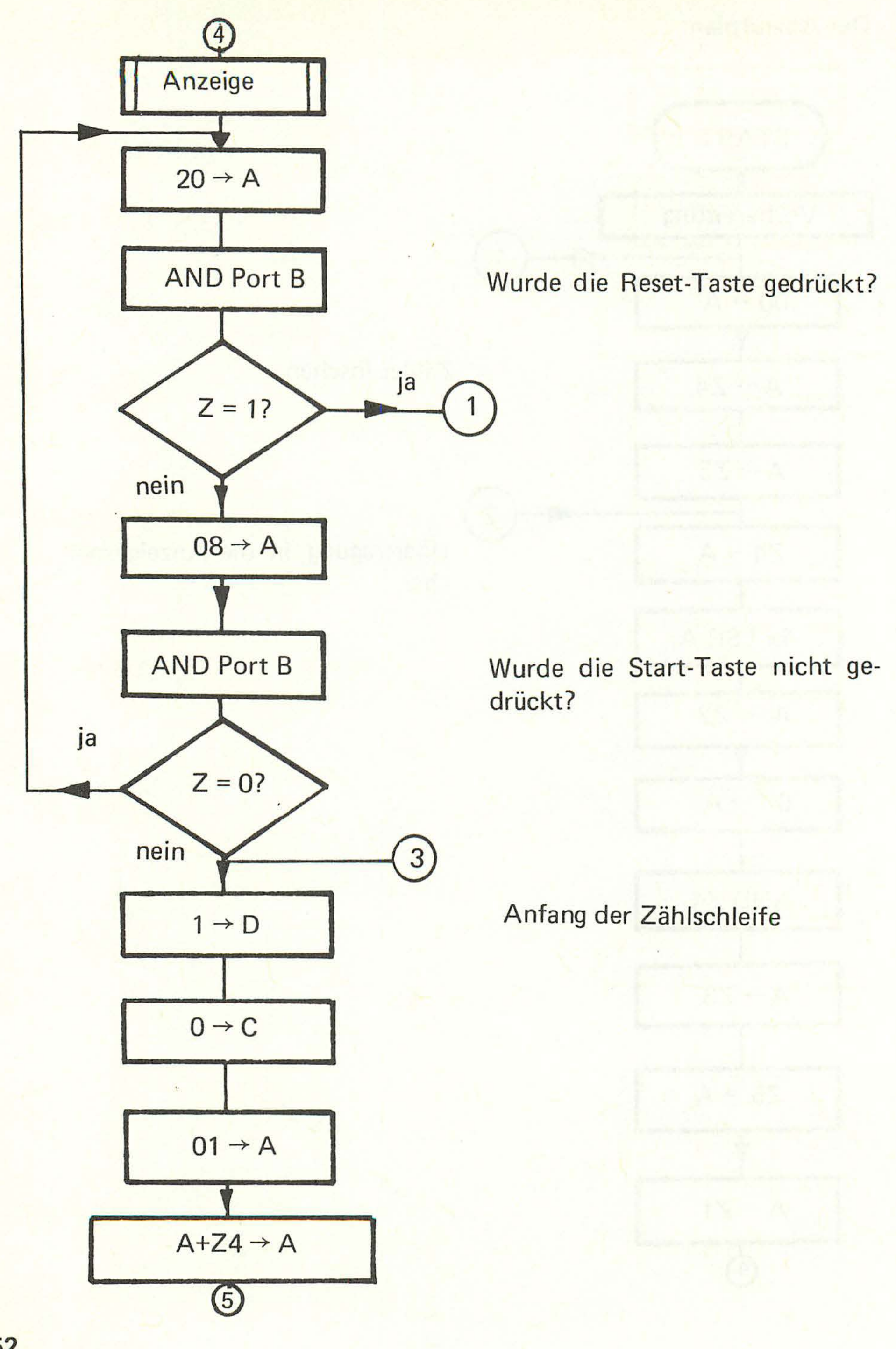
Wird die Start-Taste gedrückt, so setzt die CPU das Zählen fort. Wenn die Reset-Taste gedrückt wurde, beginnt sie das Programm von neuem mit dem Löschen der Zählerinhalte.

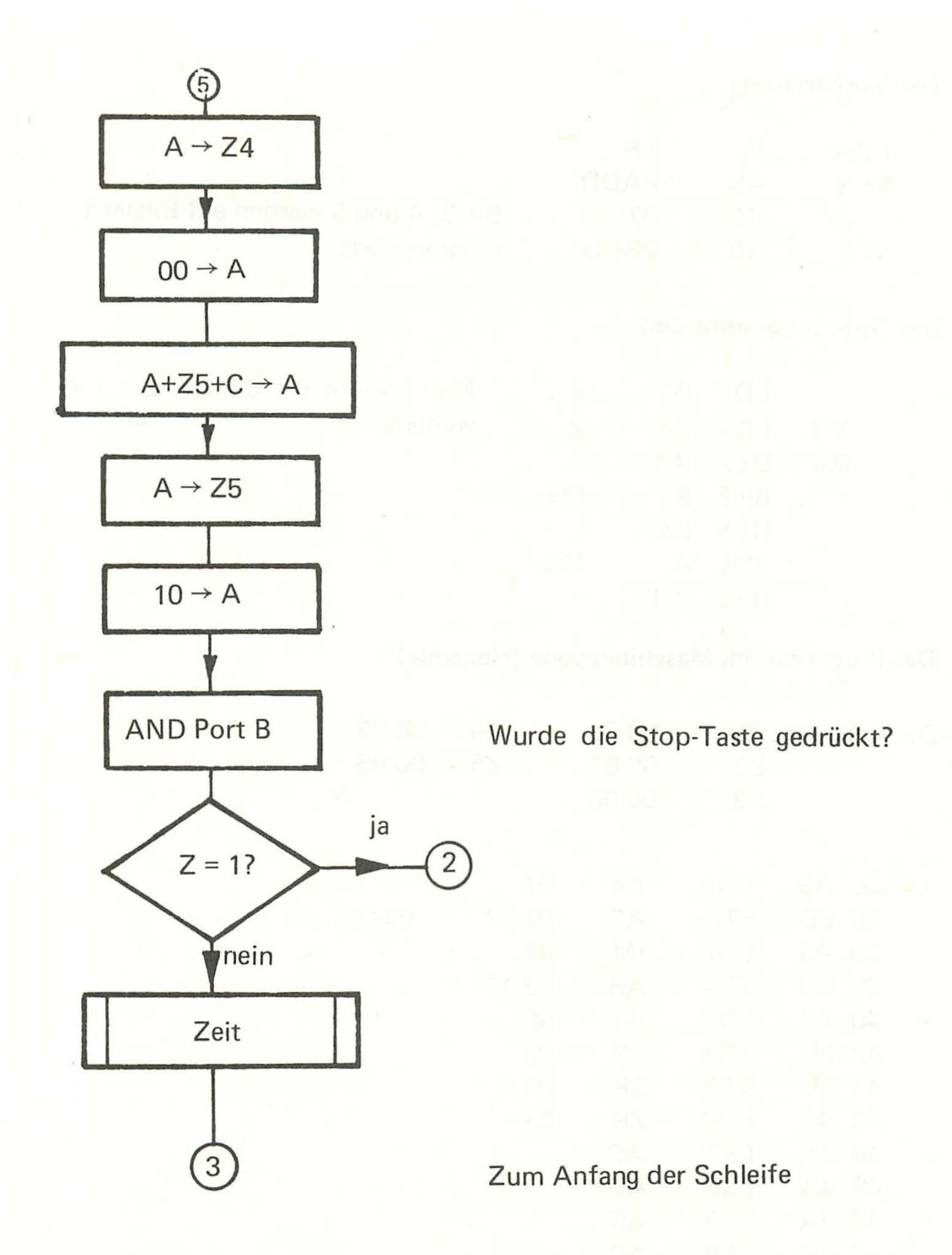
Der Ablaufplan:



Zähler löschen.

Übertragung in die Anzeigespeicher.


4
Anzeige
20 → A
AND Port B
Wurde die Reset-Taste gedrückt?
Z = 1?
ja
1
nein
08 → A
AND Port B
Wurde die Start-Taste nicht gedrückt?
ja
Z = 0?
nein
3
1 → D
Anfang der Zählschleife
0 → C
01 → A
A+Z4 → A
5



Wurde die Stop-Taste gedrückt?

Zum Anfang der Schleife

Die Vorbereitung:

LDA	IM	FF	
STA	AB	PADD	
LDA	IM	07	Bit 3, 4 und 5 werden auf Eingang
STA	AB	PBDD	programmiert.

Das Unterprogramm Zeit:

	LDY	IM	t	Man könnte t = 08 und Z = F4
M3	LDX	IM	Z	wählen.
M4	DEX	IMP		
	BNE	R	M4	
	DEY	IMP		
	BNE	R	M3	
	RTS	IMP		

Das Programm im Maschinencode (Hexcode):

Die Adressen:	Z1	00 02	Z4	00 03
	Z2	00 01	Z5	00 05
	Z3	00 00		

02 36	A9	LDA	IM	FF	
38	8D	STA	AB	01 17	PADD
3B	A9	LDA	IM	07	
3D	8D	STA	AB	03 17	
40	A9	LDA	IM	00	
42	85	STA	ZP	03	
44	85	STA	ZP	05	
46	A5	LDA	ZP	03	
48	4A	LSR	AC		
49	4A	LSR	AC		
4A	4A	LSR	AC		
4B	4A	LSR	AC		
4C	85	STA	ZP	01	
4E	A9	LDA	IM	0F	
50	25	AND	ZP	03	
52	85	STA	ZP	00	

```
54 A5   LDA   ZP   05
56 85   STA   ZP   02
58 20   JSR   AB   14 02
5B A9   LDA   IM   20
5D 2D   AND   AB   02 17
60 F0   BEQ   R    DE
62 A9   LDA   IM   08
64 2D   AND   AB   02 17
67 D0   BNE   R    F2
69 F8   SED   IMP
6A 18   CLC   IMP
6B A9   LDA   IM   01
6D 65   ADC   ZP   03
6F 85   STA   ZP   03
71 A9   LDA   IM   00
73 65   ADC   ZP   05
75 85   STA   ZP   05
77 A9   LDA   IM   10
79 2D   AND   AB   02 17
7C F0   BEQ   R    C8
7E 20   JSR   AB   84 02
81 4C   JMP   AB   69 02
```

Zeit:

```
02 84 A0   LDY   08
   86 A2   LDX   F4
   88 CA   DEX   IMP
   89 D0   BNE   R    FD
   8B 88   DEY   IMP
   8C D0   BNE   R    F8
   8E 60   RTS   IMP
```

Der Speicherauszug:

```
02 36 A9 FF 8D 01 17 A9 07 8D 03 17
02 40 A9 00 85 03 85 05 A5 03 4A 4A 4A 4A 85 01 A9 0F
02 50 25 03 85 00 A5 05 85 02 20 14 02 A9 20 2D 02 17
02 60 F0 DE A9 08 2D 02 17 D0 F2 F8 18 A9 01 65 03 85
02 70 03 A9 00 65 05 85 05 A9 10 2D 02 17 F0 C8 20 84
02 80 02 4C 69 02 A0 08 A2 F4 CA D0 FD 88 D0 F8 60
```

4. Ein Spielautomat

In vielen Gaststätten und „Spielhallen" findet man Spielautomaten mit Zufallsrädern. Wie der Spieler diese Räder geeignet stoppt, wird ein gewisser Gewinn ausgeschüttet.
Einen ähnlichen Spielautomaten wollen wir nun mit Hilfe unserer dreistelligen Anzeige programmieren.

Die Problemstellung:

Der Automat besitzt eine Start (St)-, eine Gewinnanzeige (G)- und drei Stoptasten (S1, S2 und S3).
Wird die St-Taste gedrückt, beginnen Hexziffern durch alle Anzeigen zu laufen. Wird jetzt eine beliebige S-Taste gedrückt, bleibt die zugehörige Anzeige stehen. Die beiden anderen Anzeigen laufen natürlich weiter. Drücken wir eine weitere S-Taste, bleibt die nächste Anzeige stehen und die letzte läuft noch weiter. Drücken wir die letzte S-Taste, so bleibt die letzte Anzeige stehen.
Nun soll der Gewinn berechnet werden. Die Gewinnregel lautet:

3 Anzeigen gleich:	+ 3 Punkte
2 Anzeigen gleich:	+ 1 Punkt
alle Anzeigen versch.:	– 3 Punkte

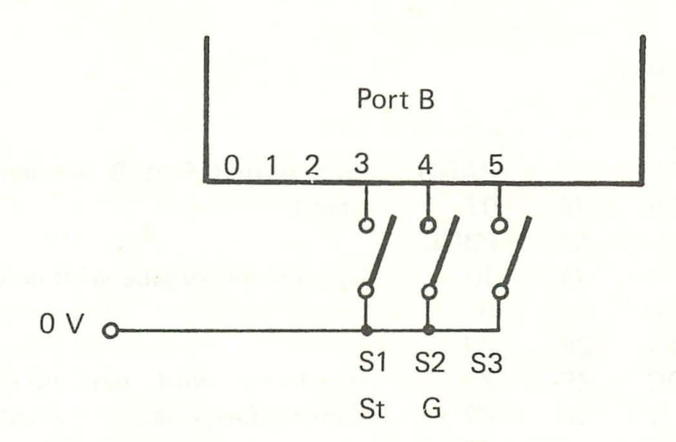
Das Spiel beginnt mit einer Vorgabe von 10 Punkten. Drücken wir, wenn alle Anzeigen stehen, die G-Taste, so folgt die Auswertung und es wird der Punktestand angezeigt.

Die Problemanalyse:

Wir müssen fünf verschiedene Tasten an einen Eingangsport anschliessen. Bei der Systemkonfiguration des KIM stoßen wir auf einige Schwierigkeiten.

Man kann sich aber auf drei zusätzliche Tasten beschränken, wenn man zwei Tasten Doppelfunktionen zuordnet. S1 erhält zusätzlich die Startfunktion und S2 die Gewinnanzeigefunktion.

Damit handeln wir uns nur den Nachteil ein, daß S2 nicht als letzte Stoptaste gedrückt werden darf. Sonst zeigt der Automat sofort den Gewinnstand an. Wir belegen den Port B des KIM also so:



Wir teilen das Programm in mehrere Teilprogramme auf. Das Unterprogramm Anzeige (im 1. Beispiel erläutert) wird ohne Änderungen übernommen. Das Hauptprogramm besteht aus zwei Teilen:

1. Zählprogramm
2. Auswertung

Natürlich wird wieder das bekannte Verzögerungsunterprogramm Zeit benutzt.
Als Datenspeicher benötigen wir die Zähler Z1, Z2, Z3 der Anzeige, zwei Hilfsspeicher M und H und den Gewinnspeicher.

Die Problemlösung:

Der 1. Teil des Programms ist eine Zählschleife, da alle Hexzeichen jede Anzeige durchlaufen sollen.
Allerdings ist diese Zählschleife etwas verzwickt aufgebaut, da beliebige Anzeigen gestoppt werden, die anderen aber weiterlaufen. Erst, wenn alle Zähler gestoppt wurden, soll die CPU die Zählschleife verlassen.

Bevor die CPU in die Zählschleife eintritt, muß sie die Zähler laden. Diese sollen nicht mit gleichen Daten geladen werden. Bei jedem neuen Spielabschnitt sollten sie möglichst mit einem anderen Inhalt beginnen.

Die Vorbereitung:

	LDA	IM	FF	
	STA	AB	PADD	Port A und Port B werden de-
	LDA	IM	07	finiert.
	STA	AB	PBDD	
	LDA	IM	10	Die Gewinnvorgabe wird gesetzt.
	STA	ZP	G	
①	LDA	ZP	Z2	
	ADC	ZP	Z3	Hierdurch wird ein zufälliger
	STA	ZP	Z2	Zähleranfangszustand erzeugt.
	SBC	ZP	Z1	
	STA	ZP	Z1	

① ist der Eintritt bei einem neuen Spieldurchgang (ohne neue Gewinnvorgabe), wenn die St-Taste gedrückt wurde.

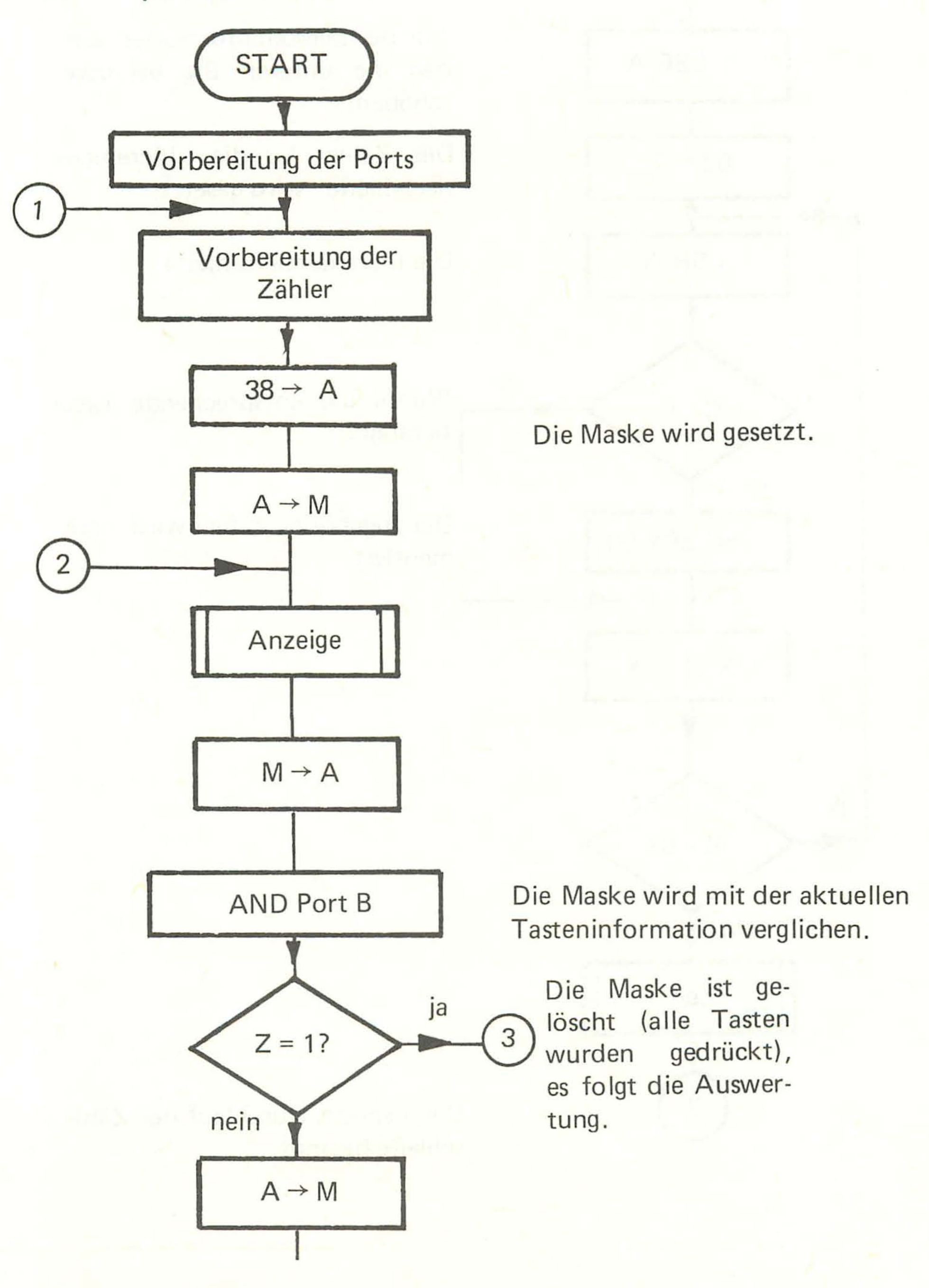
Nun wenden wir uns der Realisierung der Zählschleife zu. In den Speicher M schreibt die CPU eine Maske für die Tasten ein (00111000_2 $= 38_{16}$).

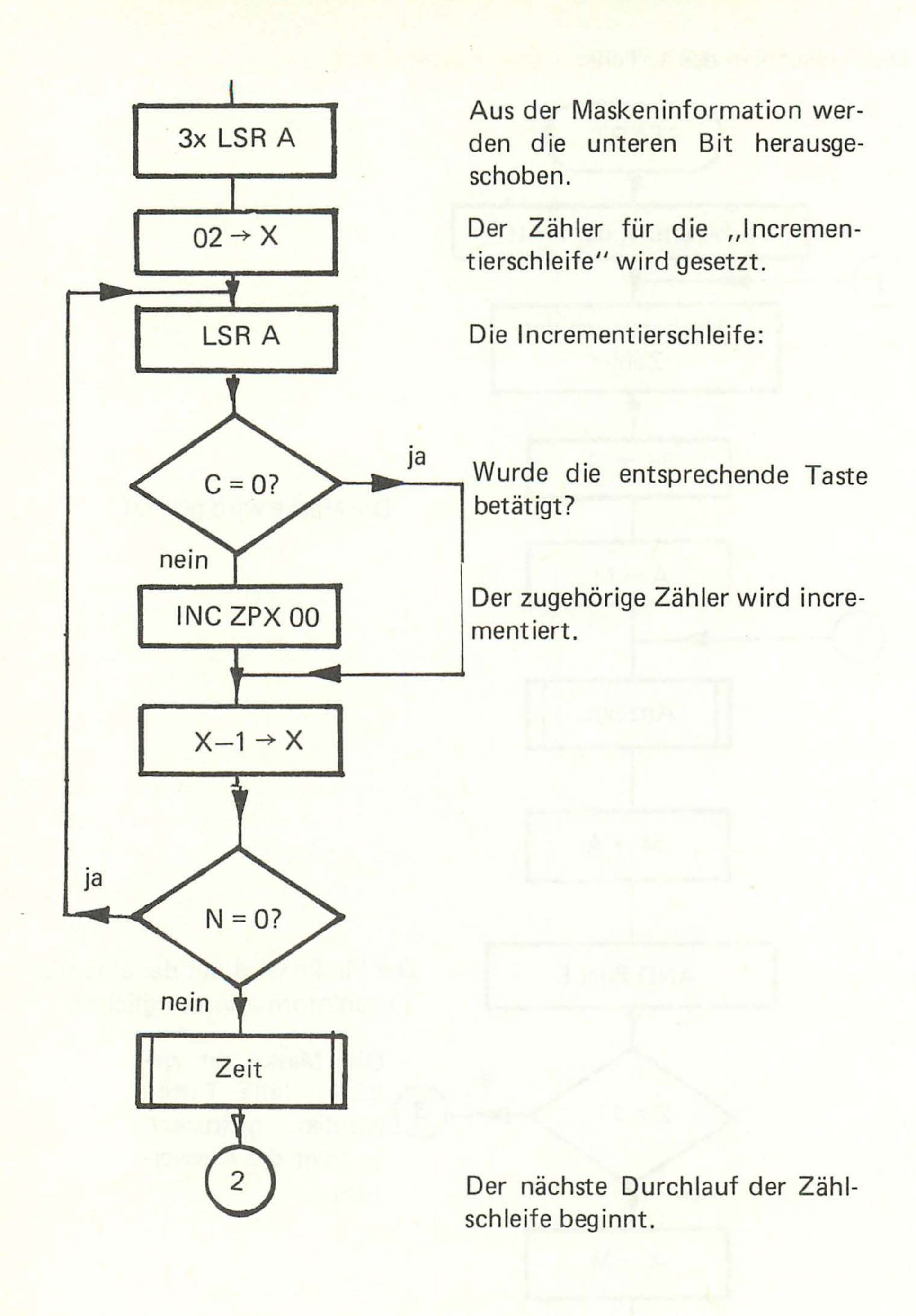
Wenn bei einem Schleifendurchgang festgestellt wurde, daß eine Taste betätigt wurde, löscht die CPU in M das entsprechende Bit. Wurden alle Tasten betätigt, ist in M der Inhalt $0000\ 0000_2$ und die CPU verläßt die Zählschleife.

Diese Maske kann die CPU beim eigentlichen Zählvorgang heranziehen, um festzustellen, welcher Zähler nicht mehr incrementiert werden soll.

Die Zähler werden als Datenfeld verstanden. In einer inneren Schleife mit drei Durchläufen und dem X-Register als Schleifenzähler und Datenzeiger wird dies besorgt.

Der Ablaufplan des 1. Teils:





Aus der Maskeninformation werden die unteren Bit herausgeschoben.

Der Zähler für die „Incrementierschleife" wird gesetzt.

Die Incrementierschleife:

Wurde die entsprechende Taste betätigt?

Der zugehörige Zähler wird incrementiert.

Der nächste Durchlauf der Zählschleife beginnt.

1. Teil im Assemblercode (ohne Vorbereitung):

```
        LDA   IM    38
        STA   ZP    M
 (2)    JSR   AB    Anzeige
        LDA   ZP    M
        AND   AB    PBD
        BEQ   R     (3)
        STA   ZP    M
        LSR   AC
        LSR   AC
        LSR   AC
        LDX   IM    02
 M1     LSR   AC
        BCC   R     M2
        INC   ZPX   00
 M2     DEX   IMP
        BPL   R     M1
        JSR   AB    Zeit
        JMP   AB    (2)
```

Wurden alle Stoptasten gedrückt, so verläßt die CPU die Zählschleife und wendet sich dem 2. Teil, der Auswertung zu. Zu Beginn der Auswertung müssen die Zähler auf eine 4 bit Breite reduziert werden. Die Anzeigen zeigen ja nur die unteren 4 Bit der Zähler an.

Dies geschieht durch die folgende Schleife. Das X-Register wird als Zähler und Zeiger eingesetzt.

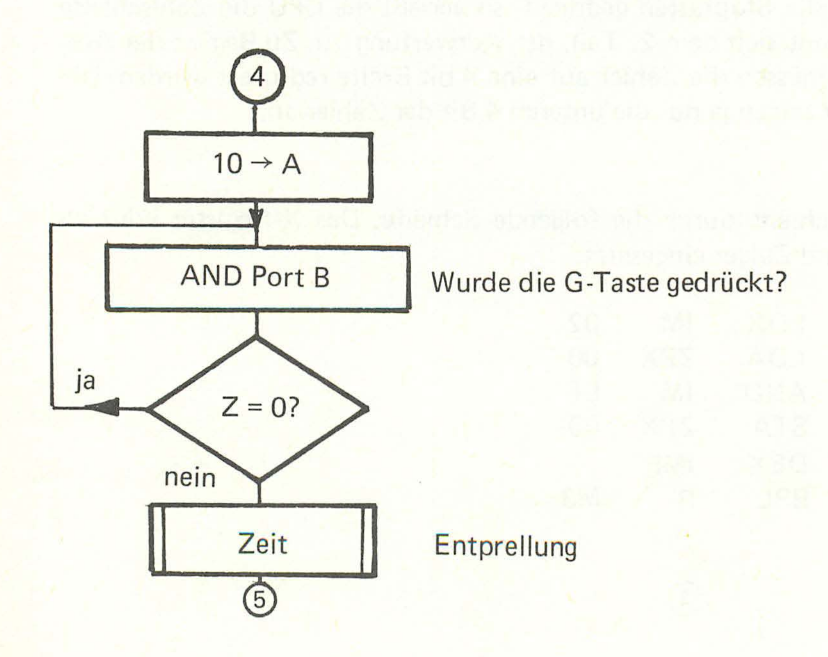
```
 (3)    LDX   IM    02
 M3     LDA   ZPX   00
        AND   IM    0F
        STA   ZPX   00
        DEX   IMP
        BPL   R     M3
              ↓
             (4)
```

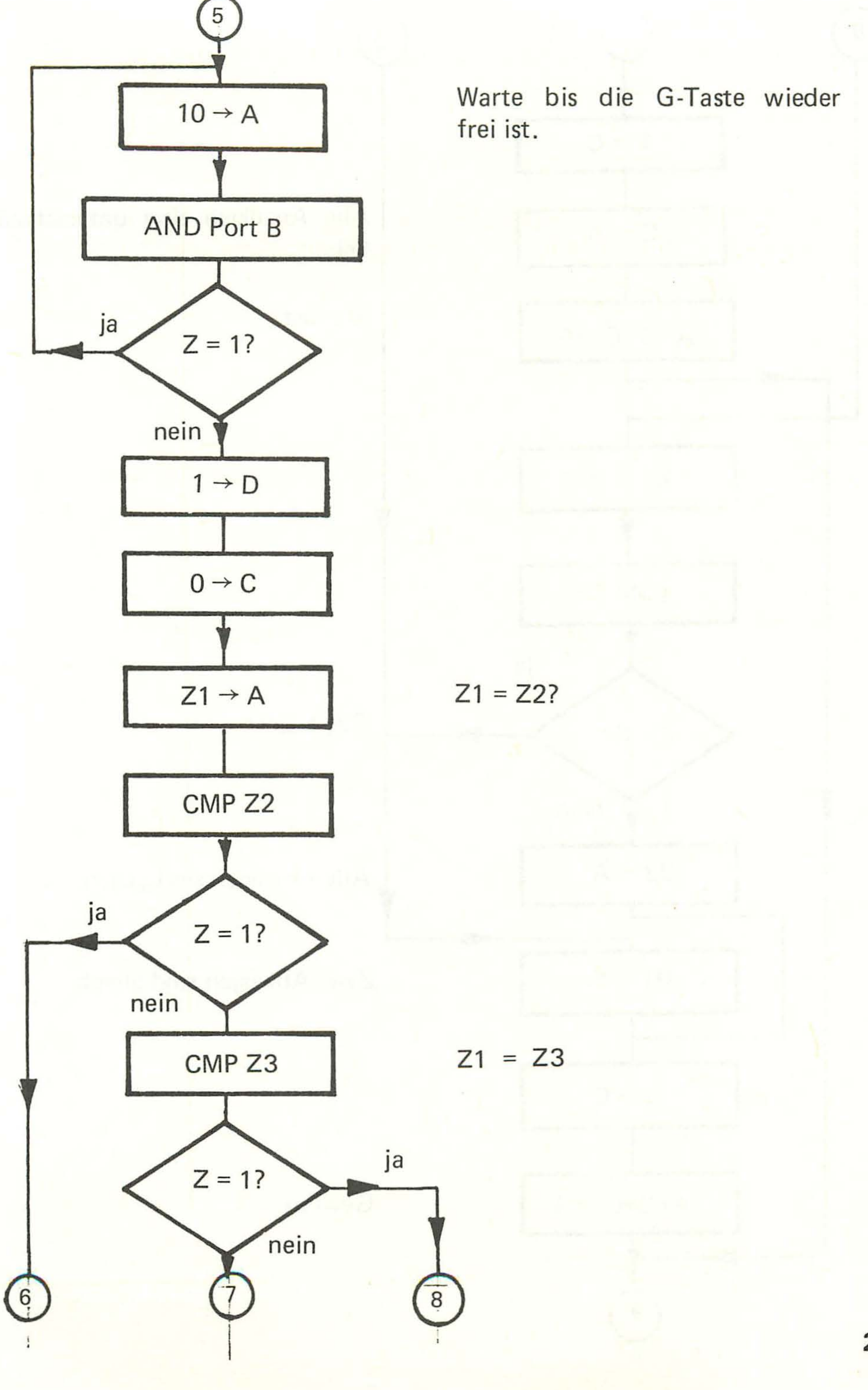
Bevor die CPU mit der Bestimmung der Punktezahlen beginnt, soll der Spieler die Möglichkeit haben, die aktuellen Zählerinhalte zu betrachten. Daher beginnt der 2. Teil mit einer Warteschleife, in der die CPU auf das Betätigen der G-Taste wartet.

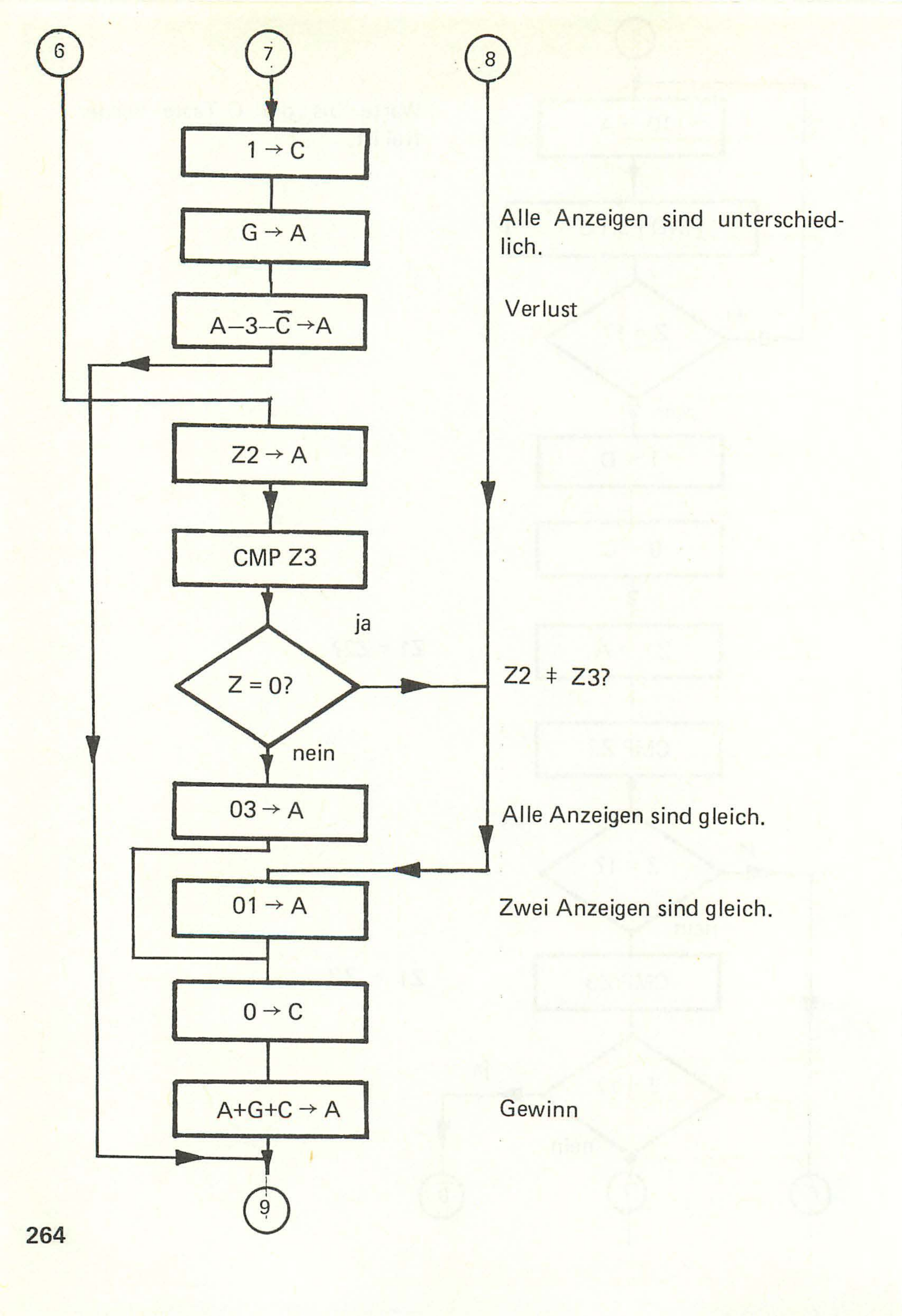
Da die Punkte im BCD-Code berechnet werden, folgt eine Umschaltung in den Dezimalmode. Anschließend werden die Zählerinhalte verglichen, um die Punktzahl zu ermitteln. Diese wird zur Gewinnzahl addiert oder von ihr subtrahiert und die Anzeigenspeicher entsprechend geladen.
Es folgt das Unterprogramm Anzeige.

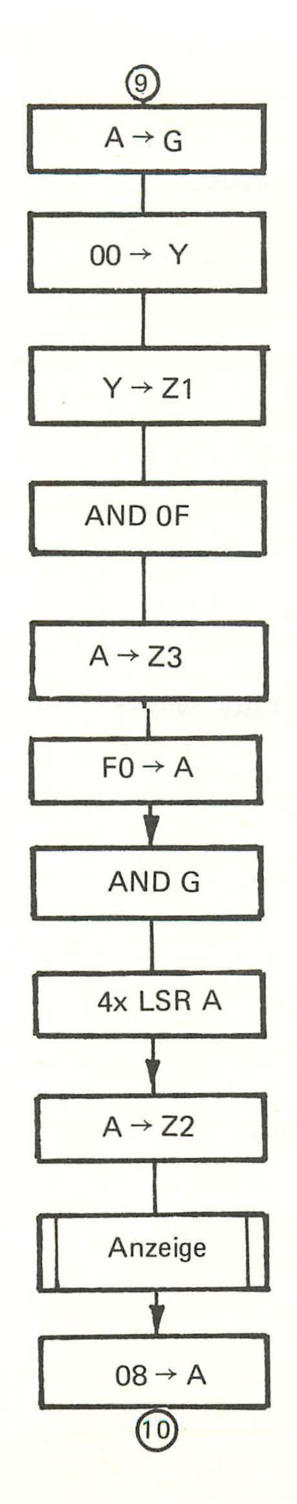
Bevor die CPU mit dem nächsten Spieldurchgang ab (1) beginnt, wartet sie in einer Warteschleife auf das Drücken der St-Taste. Sie muß für Entprellung sorgen und warten, bis die St-Taste wieder losgelassen wird, um nun an die Programmstelle (1) zu springen.

Der Ablaufplan des 2. Teils:

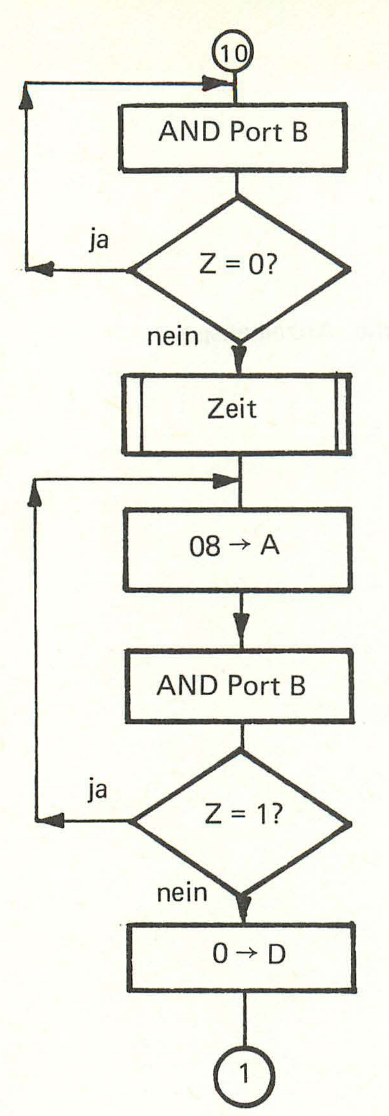



5
10 → A
AND Port B
Z = 1?
ja
nein
1 → D
0 → C
Z1 → A
CMP Z2
Z = 1?
ja
nein
CMP Z3
Z = 1?
ja
nein
6
7
8
Warte bis die G-Taste wieder frei ist.
Z1 = Z2?
Z1 = Z3


6
7
8
1 → C
G → A
A–3–C̄ → A
Z2 → A
CMP Z3
Z = 0?
ja
nein
03 → A
01 → A
0 → C
A+G+C → A
9
Alle Anzeigen sind unterschiedlich.
Verlust
Z2 ≠ Z3?
Alle Anzeigen sind gleich.
Zwei Anzeigen sind gleich.
Gewinn



Transport in die Anzeigenspeicher



Wird die St-Taste gedrückt.

Warte bis die St-Taste wieder frei ist.

Springe an den Anfang.

2. Teil im Assemblercode:

```
        LDA   IM    10
M4      AND   AB    Port B
        BNE   R     M4
        JSR   AB    Zeit
M5      LDA   IM    10
        AND   AB    Port B
        BEQ   R     M5
        SED   IMP
        CLC   IMP
        LDA   ZP    Z1
        CMP   ZP    Z2
        BEQ   R     M6
        CMP   ZP    Z3
        BEQ   R     M7
        SEC   IMP
        LDA   ZP    G
        SBC   IM    03
        JMP   AB    M9
M6      LDA   ZP    Z2
        CMP   ZP    Z3
        BNE   R     M7
        LDA   IM    03
        JMP   AB    M8
M7      LDA   IM    01
M8      CLC   IMP
        ADC   ZP    G
M9      STA   ZP    G
        LDY   IM    00
        STY   ZP    Z1
        AND   IM    0F
        STA   ZP    Z3
        LDA   IM    F0
        AND   ZP    G
        LSR   AC
        LSR   AC
        LSR   AC
        LSR   AC
        STA   ZP    Z2
```

```
        JSR   AB   Anzeige
        LDA   IM   08
M10     AND   AB   Port B
        BNE   R    M10
        JSR   AB   Zeit
M11     LDA   IM   08
        AND   AB   Port B
        BEQ   R    M11
        CLD   IMP
        JMP   AB   (1)     Anfang
```

Das Unterprogramm Zeit ist uns schon bekannt. Wir programmieren eine Doppelschleife mit den Registern X und Y als Schleifenzähler.

Das Programm im Maschinencode (Hexcode):

```
Die Adressen:        Z1     00   00
                     Z2     00   01
                     Z3     00   02
                     M      00   03
                     H      00   04
                     G      00   05
               Anzeige:     02   14
               Zeit:        02   DC
```

```
0236  a9  lda im   ff
0238  8d  sta abs  01  17
023b  a9  lda im   07
023d  8d  sta abs  03  17
0240  a9  lda im   12
0242  85  sta zp   05
0244  a5  lda zp   01
0246  65  adc zp   00
0248  85  sta zp   01
024a  e5  sbc zp   02
024c  85  sta zp   02
024e  a9  lda im   38
0250  85  sta zp   03
0252  20  jsr abs  14  02
0255  a5  lda zp   03
0257  2d  and abs  02  17
```

```
025a  f0  beq r   15
025c  85  sta zp  03
025e  4a  lsr ac
025f  4a  lsr ac
0260  4a  lsr ac
0261  a2  ldx im  02
0263  4a  lsr ac
0264  90  bcc r   02
0266  f6  inc zpx 00
0268  ca  dex imp
0269  10  bpl r   f8
026b  20  jsr abs dc  02
026e  4c  jmp abs 52  02
0271  a2  ldx im  02
0273  b5  lda zpx 00
0275  29  and im  0f
0277  95  sta zpx 00
0279  ca  dex imp
027a  10  bpl r   f7
027c  a9  lda im  10
027e  2d  and abs 02  17
0281  d0  bne r   fb
0283  20  jsr abs dc  02
0286  a9  lda im  10
0288  2d  and abs 02  17
028b  f0  beq r   f9
028d  f8  sed imp
028e  a5  lda zp  00
0290  c5  cmp zp  01
0292  f0  beq r   0c
0294  c5  cmp zp  02
0296  f0  beq r   13
0298  38  sec imp
0299  a5  lda zp  05
029b  e9  sbc im  03
029d  4c  jmp abs b0  02
02a0  a5  lda zp  01
02a2  c5  cmp zp  02
02a4  d0  bne r   05
02a6  a9  lda im  03
02a8  4c  jmp abs ad  02
02ab  a9  lda im  01
02ad  18  clc imp
```

```
02ae  65  adc zp  05
02b0  85  sta zp  05
02b2  a0  ldy im  00
02b4  84  sty zp  02
02b6  29  and im  0f
02b8  85  sta zp  00
02ba  a9  lda im  f0
02bc  25  and zp  05
02be  4a  lsr ac
02bf  4a  lsr ac
02c0  4a  lsr ac
02c1  4a  lsr ac
02c2  85  sta zp  01
02c4  20  jsr abs 14  02
02c7  a9  lda im  08
02c9  2d  and abs 02  17
02cc  d0  bne r   fb
02ce  20  jsr abs dc  02
02d1  a9  lda im  08
02d3  2d  and abs 02  17
02d6  f0  beq r   f9
02d8  d8  cld imp
02d9  4c  jmp abs 44  02
```

```
02dc  a0  ldy im  ff
02de  a2  ldx im  ff
02e0  ca  dex imp
02e1  d0  bne r   fd
02e3  88  dey imp
02e4  d0  bne r   f8
02e6  60  rts imp
```

Der Speicherauszug:

```
0200 bd 04 02 60 3f 06 5b 4f 66 6d 7d 07 7f 67 77 7c
0210 39 5e 79 71 a0 01 a2 02 86 04 b5 00 29 0f aa 20
0220 00 02 8d 00 17 98 8d 02 17 0a a8 a9 00 8d 02 17
0230 a6 04 ca 10 e3 60 a9 ff 8d 01 17 a9 07 8d 03 17
0240 a9 12 85 05 a5 01 65 00 85 01 e5 02 85 02 a9 38
0250 85 03 20 14 02 a5 03 2d 02 17 f0 15 85 03 4a 4a
0260 4a a2 02 4a 90 02 f6 00 ca 10 f8 20 dc 02 4c 52
0270 02 a2 02 b5 00 29 0f 95 00 ca 10 f7 a9 10 2d 02
0280 17 d0 fb 20 dc 02 a9 10 2d 02 17 f0 f9 f8 a5 00
0290 c5 01 f0 0c c5 02 f0 13 38 a5 05 e9 03 4c b0 02
02a0 a5 01 c5 02 d0 05 a9 03 4c ad 02 a9 01 18 65 05
02b0 85 05 a0 00 84 02 29 0f 85 00 a9 f0 25 05 4a 4a
02c0 4a 4a 85 01 20 14 02 a9 08 2d 02 17 d0 fb 20 dc
02d0 02 a9 08 2d 02 17 f0 f9 d8 4c 44 02 a0 ff a2 ff
02e0 ca d0 fd 88 d0 f8 60
```

5. Ein Glückszahlenautomat

Auch beim wöchentlichen „Tippen" kann uns unser Microcomputer helfen. Wir wollen ihn nun zu einem „Tippautomaten" programmieren. Er soll Ihnen bei den bekannten Wettspielen Toto, Lotto und Auswahlwette u. s. w. gute Dienste leisten.
Sicherlich werden Sie, wenn Sie den Computer tippen lassen, eine höhere Gewinnchance haben. Viel Glück beim Wetten!

Die Problemstellung:

Unser Computer soll, wenn er als Totogenerator geschaltet wurde, eine der Zahlen 0,1 oder 2 auswählen. Wurde er als Lottogenerator geschaltet, wählt er eine Zahl zwischen 0 und 49 aus. Bei der Auswahlwette sucht er sich eine Zahl zwischen 0 und 39 aus.
Auch andere Tippspiele soll er beherrschen.

Die Problemanalyse:

Wie kann ein solcher Zufallszahlengenerator realisiert werden?
In Kapitel 4.2 haben wir einen einfachen Zufallszahlengenerator kennengelernt. Genauso könnten wir jetzt unseren Tippautomaten programmieren.

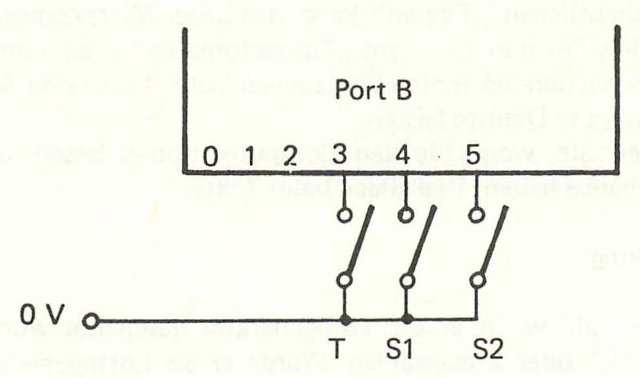
Etwas komplizierter wird die Situation aber dadurch, daß wir einen Zufallszahlengenerator für drei verschiedene Zufallszahlenbereiche zugleich programmieren wollen.

Wir entscheiden uns für die folgende Lösung.

Als zusätzliche Hardware werden ein Taster und zwei Schalter an unseren Microcomputer angeschlossen. Der Taster legt eine Zufallszahl fest, nach dem gleichen Prinzip wie in Kapitel 4.2. Die beiden Schalter können wir als eine zweistellige Dualzahl auffassen. Mit ihnen sind vier verschiedene Zahlen darstellbar. Die Schalterstellung bestimmt einen Teiler durch den die Zufallszahl dividiert wird. Der Divisionsrest ist dann der gesuchte Tippvorschlag.

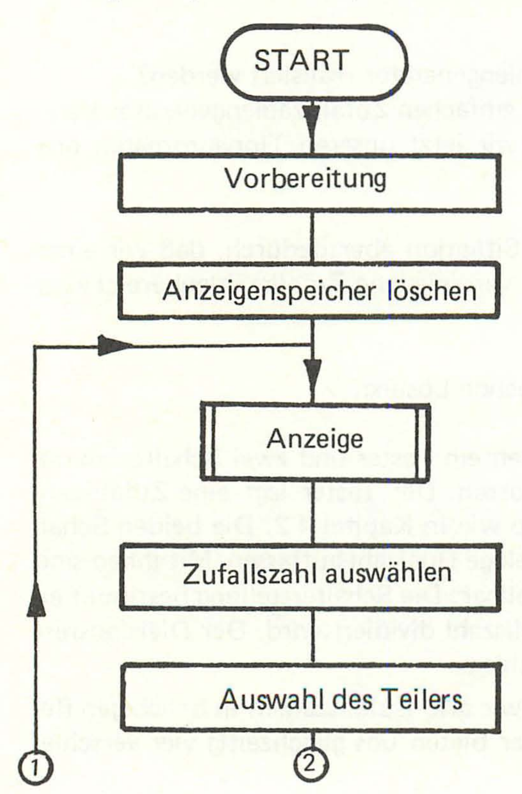
Mit diesem Generator können wir also Zufallszahlen in beliebigen Bereichen auswählen. Die Schalter bieten uns gleichzeitig vier verschiedene Möglichkeiten.

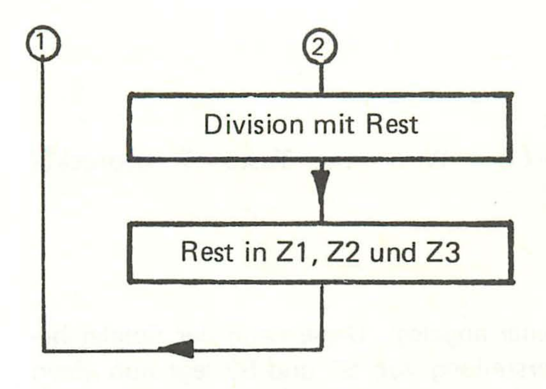
Der Taster und die beiden Schalter werden an den Port B angeschlossen.



Die Problemlösung:

Der folgende grobe Ablaufplan ist schnell aufgestellt.





Die Vorbereitung ist uns nun schon geläufig:

```
        LDA  IM  FF
        STA  AB  PADD
        LDA  IM  07
        STA  AB  PBDD
        LDA  IM  00
        STA  ZP  Z1
        STA  ZP  Z2
        STA  ZP  Z3
Anfang  JSR  AB  Anzeige
```

Wir übernehmen natürlich das uns bekannte Programm Anzeige.

Die Auswahl der Zufallszahl:

Die CPU arbeitet im Dezimalmode, da wir uns dann komplizierte Umrechnungen ersparen. Wir benutzen einen Zwischenspeicher M, in den die Zufallszahl abgelegt wird.

Die CPU bleibt so lange in einer Zählschleife, bis die Taste T gedrückt wird. Dann verläßt sie die Schleife und geht zur Bestimmung des Teilers über.

```
        SED  IMP
        LDA  IM  00
        STA  ZP  M
  M1    LDA  IM  01
```

```
ADC   ZP    M
STA   ZP    M
LDA   IM    08
AND   AB    Port B   Wird die Taste T gedrückt?
BNE   R     M1
```

Die Auswahl der Teiler:

In einem Feld T sind vier Teiler abgelegt. Diese kann der Spieler beliebig auswählen. Die Schalterstellung von S1 und S2 legt nun einen bestimmten Teiler fest. Sie wird als relative Feldadresse verstanden.

```
LDA   IM    30
AND   AB    Port B   Die Schalterinformation wird
LSR   AC             in A geladen.
LSR   AC             und die unteren Bits heraus-
LSR   AC             geschoben.
TAX   IMP            Der Zeiger wird in das X-Regi-
LDA   ABX   T        ster geladen und der Teiler ge-
BEQ   R     M3       holt. Wenn der Teiler Null ist,
                     muß die CPU die folgende Di-
                     vision überspringen.
```

Die Division mit Rest:

Es wird hier der Subtraktionsalgorithmus programmiert. Da der Quotient nicht bestimmt wird, sondern nur der Divisionsrest interessiert, ist dieser Programmteil leicht verständlich.

```
     TAY   IMP       Der Teiler kommt in das Y-Register
     LDA   ZP    M   Die Zufallszahl wird geladen
     STY   ZP    M   Der Teiler wird abgelegt.
     SEC   IMP
M2   SBC   ZP    M   Die Subtraktionsschleife
     BCS   R     M2
     ADC   ZP    M   Jetzt ist der Rest berechnet.
     STA   ZP    M   Er wird in M zwischengespeichert
```

Nun muß der Divisionsrest in die Anzeigenspeicher transportiert werden. Anschließend wartet die CPU ab, bis der Taster wieder freigegeben wird, um an den Programmanfang (M1) zurückzuspringen und die Zufallszahl über das Unterprogramm Anzeige zur Anzeige zu bringen.

```
M3    LDA   ZP   M        Wenn der Teiler 00 war, wird
      AND   IM   0F       also die Zufallszahl angezeigt.
      STA   ZP   Z1
      LDA   ZP   M
      LSR   AC
      LSR   AC
      LSR   AC
      LSR   AC
      STA   ZP   Z2
M4    LDA   IM   08
      AND   AB   Port B
      BEQ   R    M4
      JMP   AB   M1
```

Das Programm im Maschinencode (Hexcode):

```
Die Adressen:   Z1    00   00
                Z2    00   01
                Z3    00   02
                M     00   03
           Anzeige    02   14
```

```
0236  a9  lda im  ff
0238  8d  sta abs 01  17
023b  a9  lda im  07
023d  8d  sta abs 03  17
0240  a9  lda im  00
0242  85  sta zp  00
0244  85  sta zp  01
0246  85  sta zp  02
0248  20  jsr abs 14  02
024b  f8  sed imp
024c  a9  lda im  00
024e  85  sta zp  03
0250  a9  lda im  01
0252  65  adc zp  03
```

```
0254  85  sta zp  03
0256  a9  lda im  08
0258  2d  and abs 02  17
025b  d0  bne r   f3
025d  a9  lda im  30
025f  2d  and abs 02  17
0262  4a  lsr ac
0263  4a  lsr ac
0264  4a  lsr ac
0265  4a  lsr ac
0266  aa  tax imp
0267  bd  lda abx 92  02
026a  f0  beq r   0e
026c  a8  tay imp
026d  a5  lda zp  03
026f  84  sty zp  03
0271  38  sec imp
0272  e5  sbc zp  03
0274  b0  bcs r   fc
0276  65  adc zp  03
0278  85  sta zp  03
027a  a5  lda zp  03
027c  29  and im  0f
027e  85  sta zp  00
0280  a5  lda zp  03
0282  4a  lsr ac
0283  4a  lsr ac
0284  4a  lsr ac
0285  4a  lsr ac
0286  85  sta zp  01
0288  a9  lda im  08
028a  2d  and abs 02  17
028d  f0  beq r   f9
028f  4c  jmp abs 48  02
```

Der Speicherauszug:

```
0236 a9 ff 8d 01 17 a9 07 8d 03 17 a9 00 85 00 85 01
0246 85 02 20 14 02 f8 a9 00 85 03 a9 01 65 03 85 03
0256 a9 08 2d 02 17 d0 f3 a9 30 2d 02 17 4a 4a 4a 4a
0266 aa bd 92 02 f0 0e a8 a5 03 84 03 38 e5 03 b0 fc
0276 65 03 85 03 a5 03 29 0f 85 00 a5 03 4a 4a 4a 4a
0286 85 01 a9 08 2d 02 17 f0 f9 4c 48 02
```

4.7 Ein programmierbarer Musikautomat

Die Problemstellung:

In Kapitel 4.3 programmierten wir eine elektronische Orgel. Es liegt natürlich nahe, die Erfahrungen, die wir dort gewannen, zu nutzen, um eine automatische Orgel zu programmieren. Diese Orgel sollte sogar programmierbar sein. Sie muß relativ leicht auf ein anderes Lied einstellbar sein.

Man könnte mit wenig Aufwand einen Mutteroszillator entwickeln, der an den Microcomputer angeschlossen wird. Der Computer bestimmt mit Hilfe eines Programms die Tonhöhe des Mutteroszillators. Diesen Weg werden wir nicht gehen, da wir dem Prinzip dieses Buches treu bleiben und möglichst wenig zusätzliche Hardware entwickeln wollen.

Der Microcomputer soll Mutteroszillator und programmierbare Steuerung zugleich sein.

Die Problemanalyse:

Wie wir das Programm zu entwickeln haben, das den Microcomputer zu einem Mutteroszillator programmiert, ist bekannt.
Die CPU befindet sich dort in einer Schleife, die eigentlich eine sehr einfache Struktur hat. Die Frequenz des Tones (Tonhöhe) wird durch die Zeitverzögerung des Unterprogramms Zeit bestimmt.

Wenn dieser Mutteroszillator nun selbstständig ein Lied spielen soll, muß ihm pro Ton dessen Tonhöhe und dessen Länge mitgeteilt werden. Ein Musikstück besteht nicht nur aus Tönen, sondern es beinhaltet auch Pausen. Die Pausen sind ebenso wichtig wie die Töne. Sie trennen nämlich einzelne Töne und Liedabschnitte.

Eine Pause wird als Ton mit der Höhe 00 erklärt.

Die Problemlösung:

Den NF-Verstärker schließen wir an Bit 0 von Port B an.
Wie wir schon feststellten, müssen zwei Informationen pro Ton be-

kannt sein, die Tonhöhe und dessen Länge. Wir legen also zwei Datenfelder fest. FT beinhaltet die Zeitkonstanten der Töne, die der Automat beherrscht. Hier tragen wir experimentelle Konstanten ein, wenn wir den Automaten stimmen.

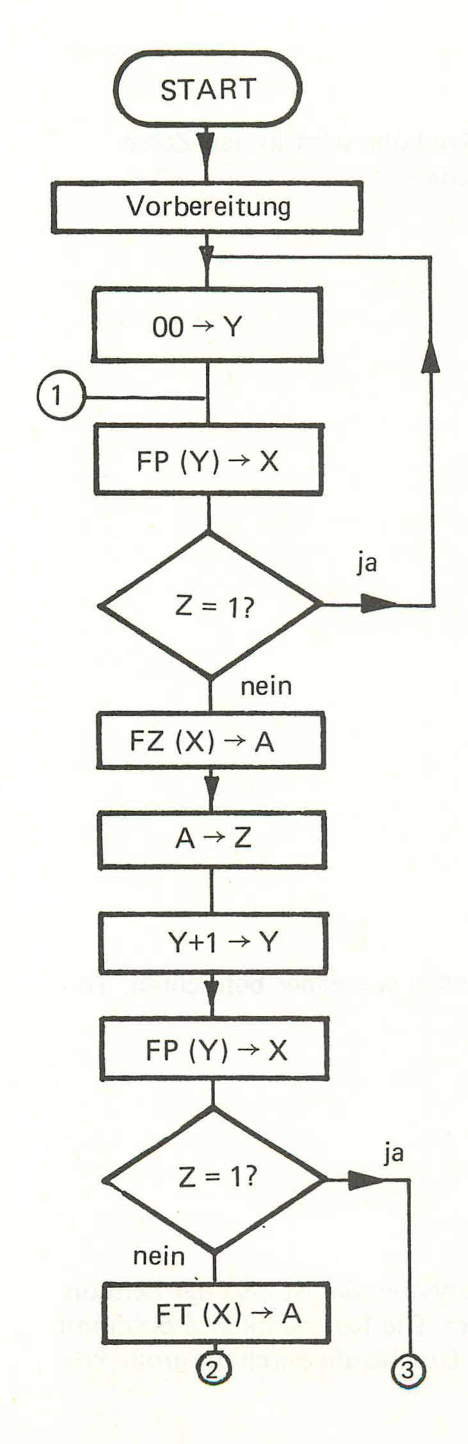
Das Feld FZ beinhaltet Konstanten, die die Länge eines Tones bestimmen. Auch hier tragen wir experimentelle Werte ein. Der Automat soll ja schnell und langsam spielen und Viertel und Achtel kennen u. s. w.

Das eigentliche „Programm", die Vorschrift, die den Automaten steuert, wenn er zum Beispiel „Hänschen klein ging allein..." spielt, wird in das Programmfeld FP abgelegt. Je zwei Bytes dieses Feldes bestimmen einen Ton. Das erste Byte gibt die Tonlänge (oder Pausenlänge) an, das zweite Byte bestimmt die Tonhöhe.

Ein Programm besteht aus einer Menge relativer Adressen, die in FP abgespeichert werden. Diese relativen Adressen weisen auf die gewünschte Tonlänge, die im Feld FZ abgespeichert ist, oder auf die gewünschte Tonhöhe, die im Feld FT abgespeichert ist.

Wie sieht nun die Struktur unseres Programmes aus?
Wir benötigen einen Zähler, der die einzelnen Programmschritte (Tonfolgen) bestimmt. Das Y-Register soll diese Aufgabe übernehmen. Wir müssen festlegen, wie das Ende eines Liedprogrammes bestimmbar sein soll. Das Byte 00 als Längenkonstante soll das Ende des Liedes markieren.

Grundsätzlich soll der Automat das Lied, wenn er sein Ende erreicht hat, wiederholen.



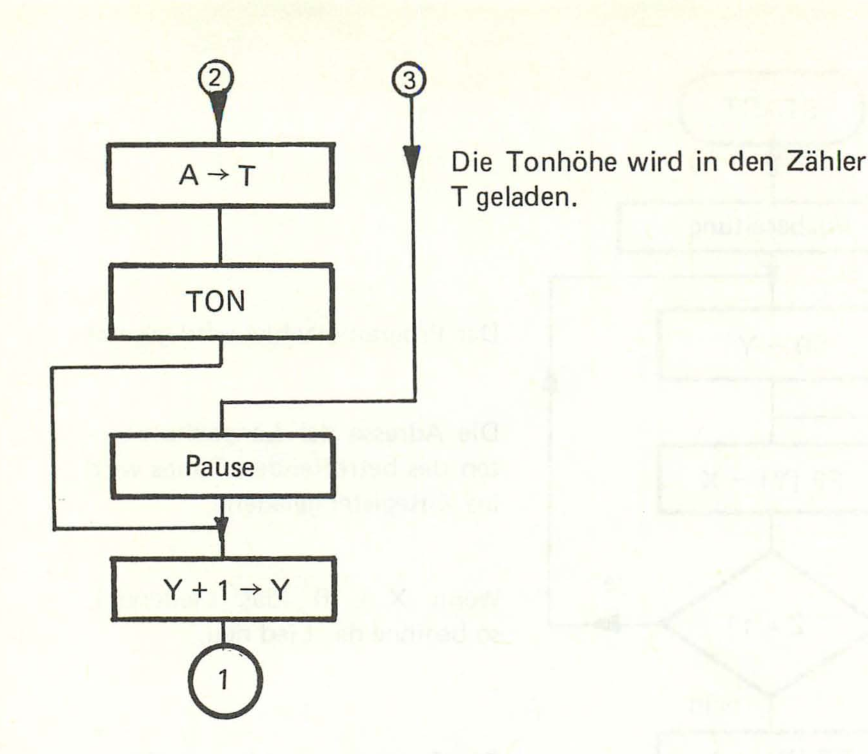
Der Programmzähler wird gesetzt

Die Adresse der Längenkonstanten des betreffenden Tones wird ins X-Register geladen.

Wenn X = 0 (das Liedende), so beginne das Lied neu.

Die Tonlänge wird in den Zähler Z geladen.

Die Adresse der Tonhöhenkonstanten des betreffenden Tones wird ins X-Register geladen.
Wenn X = 0, so liegt eine Pause vor.

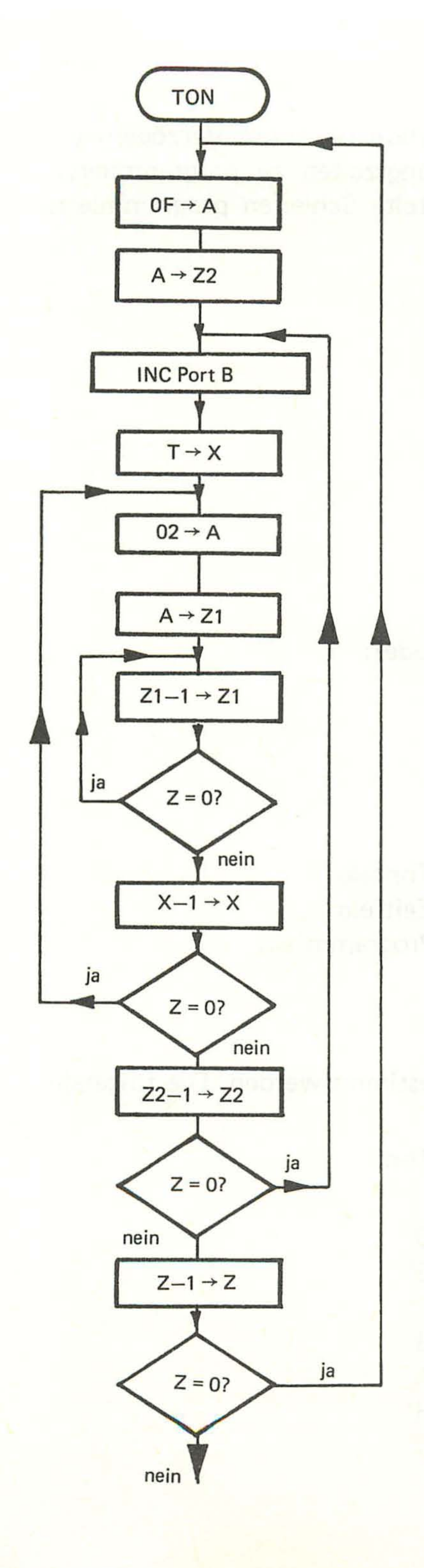


Die Tonhöhe wird in den Zähler T geladen.

Zwei Elemente dieses Programmes wollen wir näher betrachten, Ton und Pause.

Das Programmelement Ton:

Die Konstante in T bestimmt die Tonhöhe. Sie ist also die Zeitkonstante des Verzögerungsprogrammteiles. Die Konstante in Z bestimmt die Tonlänge. Sie gibt die Anzahl der Durchläufe durch die große Programmschleife an.



Der Zähler wird gesetzt.

Bit 0 von Port B wechselt.

Die Verzögerungsdoppelschleife

Abfrage des Z-Flag
Wenn der Tonlängenzähler = 0
wird, ist der Ton zu beenden.

Das Programmelement Pause:

Dieses Programmelement ist das uns schon bekannte Verzögerungsprogramm. Da relativ große Verzögerungszeiten zu programmieren sind, wurden drei ineinander verschachtelte Schleifen programmiert.

```
M6   LDA  IM  FF
     STA  AB  Z1
M7   LDA  IM  FF
     STA  AB  Z2
M8   DEC  AB  Z2
     BNE  R   M8
     DEC  AB  Z1
     BNE  R   M7
     DEC  AB  Z
     BNE  R   M6
```

Das Programm im Maschinencode (Hexcode):

Die Adressen:	Z	02 00
	Z1	02 01
	Z2	02 02

FT	00 00	bis	00 1F	Tonfeld
FZ	00 20	bis	00 3F	Zeitfeld
FP	00 40	bis	00 EF	Programmfeld

Einige Bemerkungen zum Probelauf:

Die Tonhöhen müssen experimentell bestimmt werden. Die folgende Tabelle kann Anhaltspunkte bieten.

FT	Ton	FT	Ton
60	C		
5A	D	2D	D'
50	E	28	E'
48	F	24	F'
40	G	20	G'
3C	A	1E	A'
36	H	1B	H'
30	C'	18	C''

Als Anhaltspunkt für die Zeiteinheit kann etwa das Byte 10 angegeben werden.

Bevor Sie sich ein Lied für Ihre „Spieldose“ aussuchen, tragen Sie die doppelte Oktave in FT ein und bestimmen Sie geeignete Werte für die Tonlängenkonstanten. Hierzu könnten Sie in das Programmfeld FP sehr kurze Programme, die nur aus zwei Anweisungen bestehen, eintragen.

Das Programm im Maschinencode (Hexcode):

```
0203  a9  lda im   01
0205  8d  sta abs  03  17
0208  a0  ldy im   00
020a  b6  ldx zpy  40
020c  f0  beq r    fa
020e  b5  lda zpx  20
0210  8d  sta abs  00  02
0213  c8  iny imp
0214  b6  ldx zpy  40
0216  f0  beq r    29
0218  b5  lda zpx  00
021a  8d  sta abs  26  02
021d  a9  lda im   0f
021f  8d  sta abs  02  02
0222  ee  inc abs  02  17
0225  a2  ldx im   30
0227  a9  lda im   02
0229  8d  sta abs  01  02
022c  ce  dec abs  01  02
022f  d0  bne r    fb
0231  ca  dex imp
0232  d0  bne r    f3
0234  ce  dec abs  02  02
0237  d0  bne r    e9
0239  ce  dec abs  00  02
023c  d0  bne r    df
023e  4c  jmp abs  5a  02
0241  a9  lda im   0f
0243  8d  sta abs  01  02
0246  a9  lda im   ff
```

```
0248  8d  sta abs 02  02
024b  ce  dec abs 02  02
024e  d0  bne r   fb
0250  ce  dec abs 01  02
0253  d0  bne r   f1
0255  ce  dec abs 00  02
0258  d0  bne r   e7
025a  c8  iny imp
025b  4c  jmp abs 0a  02
```

Der Speicherauszug:

```
0203 a9 01 8d 03 17 a0 00 b6 40 f0 fa b5 20 8d 00 02
0213 c8 b6 40 f0 29 b5 00 8d 26 02 a9 0f 8d 02 02 ee
0223 02 17 a2 30 a9 02 8d 01 02 ce 01 02 d0 fb ca d0
0233 f3 ce 02 02 d0 e9 ce 00 02 d0 df 4c 5a 02 a9 0f
0243 8d 01 02 a9 ff 8d 02 02 ce 02 02 d0 fb ce 01 02
0253 d0 f1 ce 00 02 d0 e7 c8 4c 0a 02
```

Ein Beispiel:	1. Anweisung:	Tonlänge:	01
		Tonhöhe:	01
	2. Anweisung:	Tonlänge:	01
		Pause:	00
		Ende:	00

Nun können Sie das erste Byte des Feldes FZ variieren und die entsprechenden Konstanten erproben.
Selbstverständlich können Sie mit solch einfachen Programmen den Automaten auch umstimmen. Sie können Halbtonschritte einfügen u. a. m..

Wenn Sie ein Lied einprogrammieren, sollten Sie darauf achten, zwischen jeden einzelnen Ton eine kurze Pause vorzusehen, denn sonst würden alle Töne ohne Absatz aufeinander folgen.

Dieser Effekt ist nur in manchen Fällen erwünscht (gebundene Noten).

4.8 Die Verwendung eines 5 Kanal Fernschreibers als Datenstation

Eine alphanumerische Datenstation erweitert das Spektrum der Anwendungsmöglichkeiten eines kleinen Microcomputers schon wesentlich, da dem Anwender die vielfältigen Möglichkeiten der Textverarbeitung zur Verfügung stehen.

Besitzt der Anwender eine druckende Datenstation, kann er seine Programme auslisten lassen und dann diese einfacher dokumentieren.

Dem erfahrenen Programmierer wird die Programmierung im Maschinencode sicherlich auf die Dauer lästig und er wird sich nach einem geeigneten Assemblerprogramm für seinen Microcomputer umsehen. Solche Programme sind für fast alle Microprozessoren auf dem Softwaremarkt erhältlich. Er kann natürlich auch seinen Assembler selber entwerfen. Für den Anfänger ist dies natürlich zu kompliziert.

Um mit einem Assembler arbeiten zu können, benötigt der Anwender allerdings eine alphanumerische Datenstation.

Die Textverarbeitung ist auch ein sehr interessantes Gebiet der Microcomputerprogrammierung. Sie bietet eine Unmenge praktischer Anwendungsmöglichkeiten. Man kann bestimmte Standardtexte (Geschäftsfernschreiben) abspeichern und ausdrucken. Man kann sich eine einfache Adressdatei aufbauen. Auch viele eindrucksvolle Spiele mit Textvariablen könnten realisiert werden.

Eine einfache Anwendung, die auch für den Anfänger nicht zu kompliziert ist, wird im 5. Abschnitt dieses Kapitels vorgestellt.

Nun sind die Datenstationen, die für die Microcomputer geeignet sind, recht teuer.

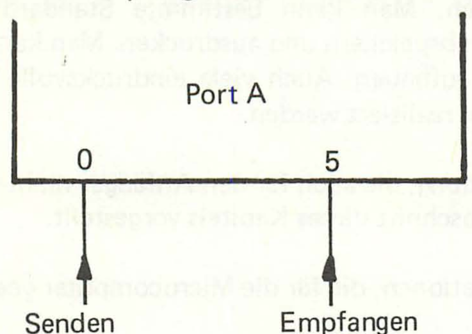
Die Betriebssysteme der Microcomputer sind auf den ASCII-Code ausgerichtet und die Schnittstellen (Interface) auf die amerikanische Fernschreibnorm (20 mA – TTY) zugeschnitten. Wenn der Anwender ein anderes Gerät anschließen will, muß er eine eigene Treibersoftware schreiben und das entsprechende Interface selber entwickeln.

Eine Datenstation, die in Amateurfunkerkreisen Verbreitung gefunden hat und zur Zeit recht preisgünstig aus Restbeständen und Ausmusterungslagern erstanden werden kann, ist der 5-Kanal-Fernschreiber. Dieser Fernschreiber ist nicht kompatibel mit einer ASCII-TTY-Station. Daher müssen wir ein spezielles Interface entwickeln. Im 1. Abschnitt wird eine einfache Möglichkeit vorgestellt. Die für dieses Interface notwendigen Treiberprogramme lernen wir in den beiden folgenden Abschnitten kennen. Der 4. Abschnitt stellt einige Testprogramme vor, mit denen wir unser Interface und die Treiberprogramme testen. Eine sinnvolle Anwendung bietet der 6. Abschnitt. Wir entwickeln dort ein Programm, das bestimmte Speicherbereiche auslistet.

1. Das Interface

Die Datenübertragung zwischen Computer und Fernschreiber geschieht seriell. Es muß also eine Wandlung von paralleler Form in die serielle und umgekehrt stattfinden. Hierfür gibt es spezielle Bausteine (sogenannte USART's). Man kann aber auch auf solche Bausteine verzichten und eine Softwarewandlung programmieren. Die Schnittstelle wird dann wesentlich einfacher. Wir wollen diesen Weg gehen.

Die Kommunikation findet über zwei Kanäle statt, einen Empfangskanal und einen Sendekanal. Der Ausgangsport A wird hierzu herangezogen. Bit 5 sei der Empfangskanal und Bit 0 sei der Sendekanal.



Wie werden nun diese Kanäle an die 40 mA Stromschleife des Fernschreibers angeschlossen?
Der Fernschreiber besteht aus zwei getrennten Elementen, dem Drucker, an dem meist auch ein Lochstreifenstanzer angeschlossen ist, und dem Tastenfeld.

Abb. 4.8.1

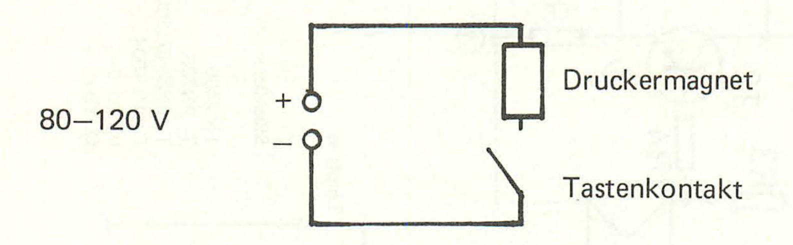
Abb. 4.8.2

Stückliste:

1 x 7400	R_1 3,8k	C_1 0,1M
T_1 2N706	R_2 56	C_2 4,7M
T_2 2N3440 (300V)	R_3 22k	
D 1N 4004	R_4 22k	
OK_1 u. OK_2	R_5 100	
CQY80	R_6 100	
	R_7 1k	
	R_8 680	

Ein eventuell vorhandener Lochstreifenleser wäre das dritte Element.

Will man den Fernschreiber als Schreibmaschine einsetzen, muß man den Druckermagneten und den Tastenfeldkontakt, der die Impulse erzeugt, mit einem entsprechend dimensionierten Linienstromnetzgerät in Serie schalten. (Die Linienstromschleife schließen). Die Dimensionierung des Netzgerätes hängt von der verwendeten Maschine ab. Es sollte im Ruhezustand ein Strom von 40mA fließen.



Wird eine Taste gedrückt, so wird der Stromkreis dem Code des Zeichens entsprechend unterbrochen. Diese Impulse steuern das Druckwerk, das das Zeichen druckt.

Soll unser Computer ein Zeichen senden, so muß der Ausgangskanal für eine dem Code entsprechende Unterbrechung des Linienstromkreises sorgen.

Abbildung 4.8.1 zeigt das Interface für den Sendekanal. Der Transistor T_2 ist in den Linienstromkreis eingefügt. Er unterbricht den Linienstrom im Rythmus der gesendeten Impulse. Die Diode D und das R–C–Glied C_1, R_8 schützen den Schalttransistor vor Überspannungen. Am Widerstand R_5 entsteht ein Spannungsabfall, wenn der Stromkreis geschlossen ist. Am Punkt A wird der Empfangskanal angeschlossen.

Ein Optokoppler trennt den Schalttransistor vom Sendekanal. Dadurch wird der Computer vor eventuell auftretenden Induktionsspannungen im Linienstromkreis geschützt. Der Transistor T_1 steuert den Optokoppler. Die beiden Nandgatter werden an den Ausgangskanal angeschlossen. Mit S_1 kann man zwischen direktem und indirektem Betrieb wählen.

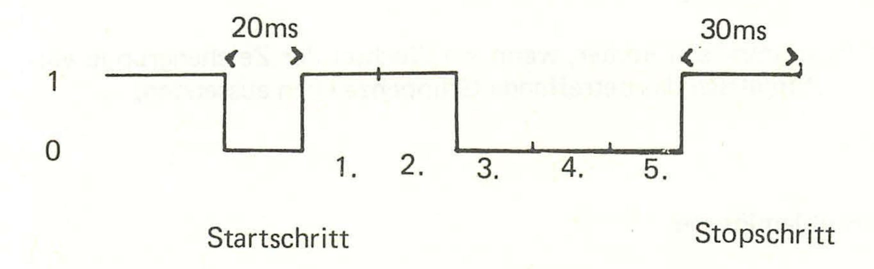
Das Empfangsinterface ist in Abbildung 4.8.2 wiedergegeben. Die Impulse, die an A entstehen, werden auf den Optokoppler geleitet. Dieser steuert wiederum zwei Nandgatter. Mittels S_2 kann zwischen direktem und indirektem Betrieb gewählt werden.

2. Das Sendeprogramm

Wie sieht der zu übertragende Code eines Zeichens aus?

In Kapitel 2.1.2 ist die Codetabelle des Fernschreibcodes zu finden. Das Zeichen A besitzt zum Beispiel den Code 00011. Eine 1 heißt, der Stromkreis ist geschlossen, eine 0 heißt, er ist unterbrochen. Die Zeichen werden von rechts nach links gesendet. Wird ein Zeichen gesendet, muß vorher eine Marke kommen, die den Zeichenanfang kennzeichnet. Am Zeichenende muß eine Marke folgen, die das Zeichen beendet.

Da die Bits seriell übertragen werden, muß jedes Bit eine bestimmte Zeit (20 ms) auf der Leitung sein, bevor das nächste Bit kommt.



Hätte der 5 Kanalcode für jedes Zeichen einen eigenen Code, wäre das Sendeprogramm relativ einfach zu entwickeln. Die unglückliche Trennung der Zeichen in zwei Gruppen (1. Gruppe: Ziffern und Zeichen; 2. Gruppe: Buchstaben), die den gleichen Code besitzen, verkompliziert das Sendeprogramm allerdings.

Jede Gruppe besitzt ihr Steuerzeichen. Sollen Buchstaben gedruckt werden, muß erst das Steuerzeichen A... gesendet werden, dann können die Buchstaben folgen. Kommen anschließend einige Ziffern, muß vorher das Steuerzeichen 1... gesendet werden.

Es wäre eine unnötige Verlangsamung der Druckergeschwindigkeit (die sowieso nur ca. 50 Baud beträgt), wenn wir vor jedem Zeichen grundsätzlich das zugehörige Gruppenzeichen senden ließen. Ein solches Programm wäre auch nicht sehr schwer zu entwickeln.

Wir wollen uns die Arbeit nicht so leicht machen und ein Sendeprogramm entwickeln, das die Sonderzeichen nur beim Zeichenwechsel automatisch in den Zeichenstrom einfügt.

Der 5 bit Code wird zu einem 6 bit Code erweitert, damit das Programm bestimmen kann, wann Buchstaben vorliegen (6. Bit = 0) oder Ziffern (6. Bit = 1). Die entsprechende Tabelle ist in Kapitel 4.1.3 zu finden.

Die Problemstellung:

Das Sendeprogramm soll ein Zeichen, das im 6 bit Code vorliegt, über den Ausgabekanal seriell senden, indem es einen Startschritt und zwei Stopschritte einfügt.

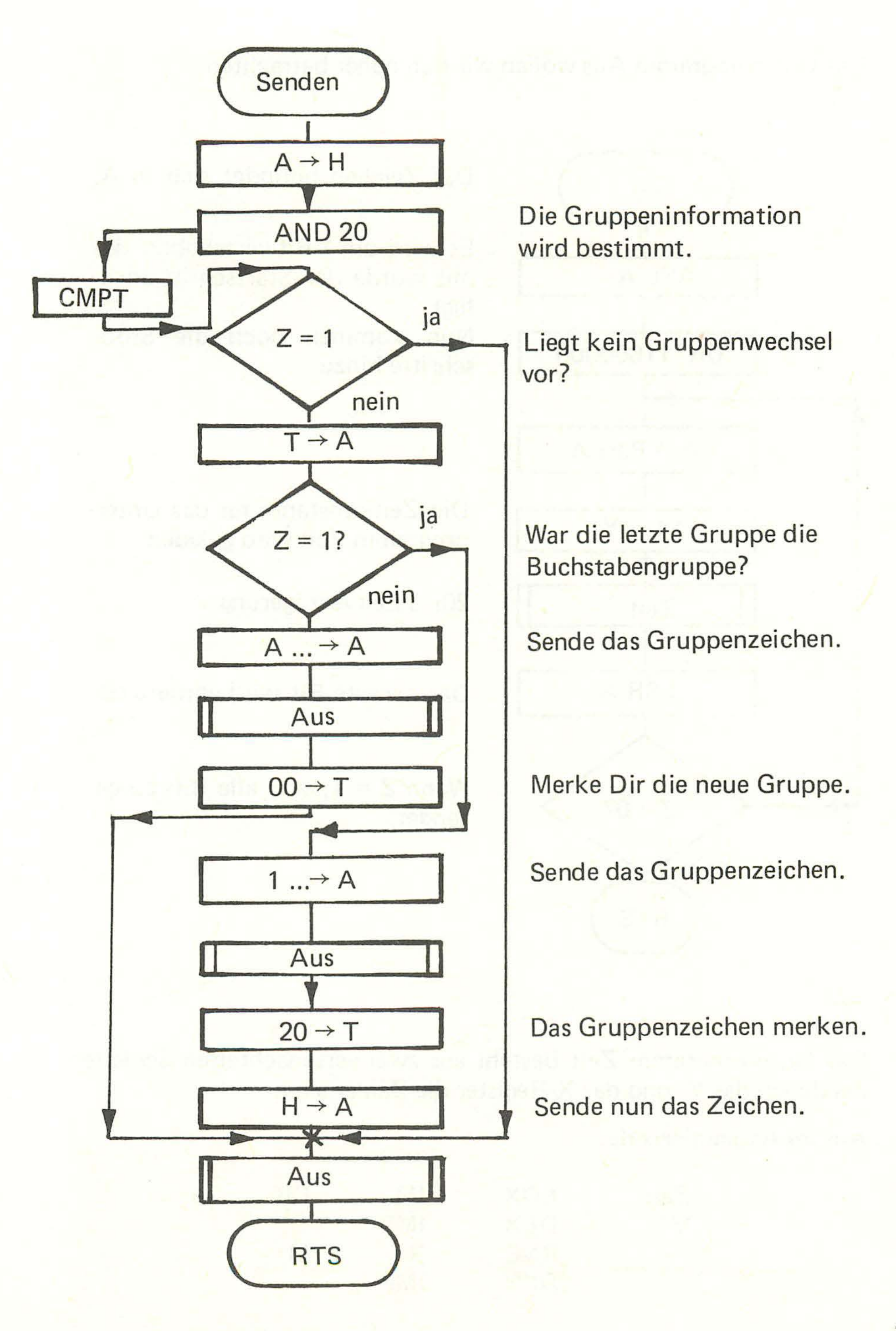
Das Programm soll immer, wenn ein Wechsel der Zeichengruppe vorliegt, automatisch das betreffende Gruppenzeichen aussenden.

Die Problemlösung:

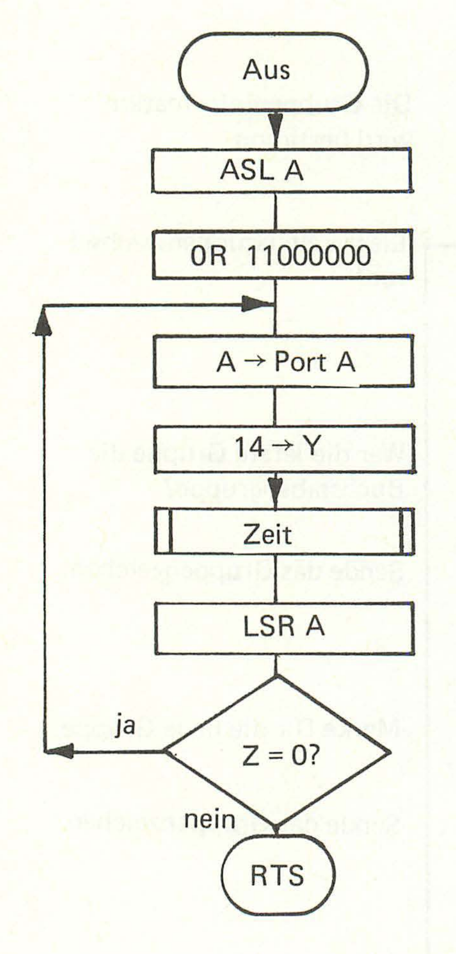
Das zu sendende Zeichen befindet sich im Akkumulator. Die CPU muß nun testen, ob dessen Gruppe gleich der Gruppe des letzten Zeichens ist. Wenn dies der Fall ist, kann sie es direkt senden. Trifft es nicht zu, muß sie die Gruppenzugehörigkeit bestimmen, das Gruppenzeichen senden und sich die neue Gruppe merken. Erst dann kann sie das eigentliche Zeichen aussenden.

Wir benötigen zwei Zwischenspeicher: H, der das Zeichen aufnimmt, und T, der die Gruppenzugehörigkeit speichert. Das Byte 20 bedeutet Ziffergruppe. Das Byte 00 bedeutet Buchstabengruppe.

Das Sendeprogramm wird natürlich als Unterprogramm gestaltet.


Senden
A → H
AND 20
Die Gruppeninformation wird bestimmt.
CMPT
Z = 1
ja
nein
Liegt kein Gruppenwechsel vor?
T → A
Z = 1?
ja
nein
War die letzte Gruppe die Buchstabengruppe?
A ... → A
Sende das Gruppenzeichen.
Aus
00 → T
Merke Dir die neue Gruppe.
1 ...→ A
Sende das Gruppenzeichen.
Aus
20 → T
Das Gruppenzeichen merken.
H → A
Sende nun das Zeichen.
Aus
RTS

Das Unterprogramm Aus wollen wir nun näher betrachten.



Das Zeichen befindet sich in A.

Es wird um 1 Bit verschoben, damit wurde der Startschritt angefügt.
Nun kommen noch die Stopschritte hinzu.

Die Zeitkonstante für das Unterprogramm Zeit wird geladen.

20ms Zeitverzögerung

Das nächste Bit wird vorbereitet.

Wenn Z = 1, sind alle Bits ausgesendet.

Das Unterprogramm Zeit besteht aus zwei verschachtelten Schleifen, bei denen das Y- und das X-Register die Zähler sind.

Aus im Assemblercode:

```
Zeit    LDX    IM     CB
M1      DEX    IMP
        BNE    R      M1
        DEY    IMP
```

	BNE	R	Zeit
	RTS	IMP	
Aus	ASL	AC	
	ORA	IM	C0
M2	STA	AB	Port A
	LDY	IM	14
	JSR	AB	Zeit
	LSR	AC	
	BNE	R	M2
	RTS	IMP	

Senden im Assemblercode:

	STA	ZP	H
	AND	IM	20
	CMP	ZP	T
	BEQ	R	V1
	LDA	ZP	T
	BEQ	R	V2
	LDA	IM	A...
	JSR	AB	Aus
	LDA	IM	00
	STA	ZP	T
	JMP	AB	V1
V2	LDA	IM	1...
	JSR	AB	Aus
	LDA	IM	20
	STA	ZP	T
V1	LDA	ZP	H
	JSR	AB	Aus
	RTS	IMP	

Das Programm im Maschinencode (Hexcode):

Die Adressen:

T	00 09
H	00 0A
Zeit	02 00
Aus	02 09
Senden	02 18

02	00	A2	LDX	IM	CB
	02	CA	DEX	IMP	
	03	D0	BNE	R	FD

```
05  88   DEY  IMP
06  D0   BNE  R     F8
08  60   RTS  IMP
09  0A   ASL  AC
0A  09   ORA  IM    C0
0C  8D   STA  ABS   00 17
0F  A0   LDY  IM    14
11  20   JSR  AB    00 02
14  4A   LSR  AC
15  D0   BNE  R     F5
17  60   RTS  IMP
18  85   STA  ZP    0A
1A  29   AND  IM    20
1C  C5   CMP  ZP    09
1E  F0   BEQ  R     09
20  A5   LDA  ZP    09
22  F0   BEQ  R     0C
24  A9   LDA  IM    1F
26  20   JSR  AB    09 02
29  A9   LDA  IM    00
2B  85   STA  ZP    09
2D  4C   JMP  AB    39 02
30  A9   LDA  IM    1B
32  20   JSR  AB    09 02
35  A9   LDA  IM    20
37  85   STA  ZP    09
39  A5   STA  ZP    0A
3B  20   JSR  AB    09 02
3E  60   RTS  IMP
```

Der Speicherauszug:

```
0200  A2 CB CA D0 FD 88 D0 F8 60 0A 09 C0 8D 00 17 A0
0210  14 20 00 02 4A D0 F5 60 85 0A 29 20 C5 09 F0 09
0220  A5 09 F0 0C A9 1F 20 09 02 A9 00 85 09 4C 39 02
0230  A9 1B 20 09 02 A9 20 85 09 A5 0A 20 09 02 60
```

3. Das Empfangsprogramm

Die Problemstellung:

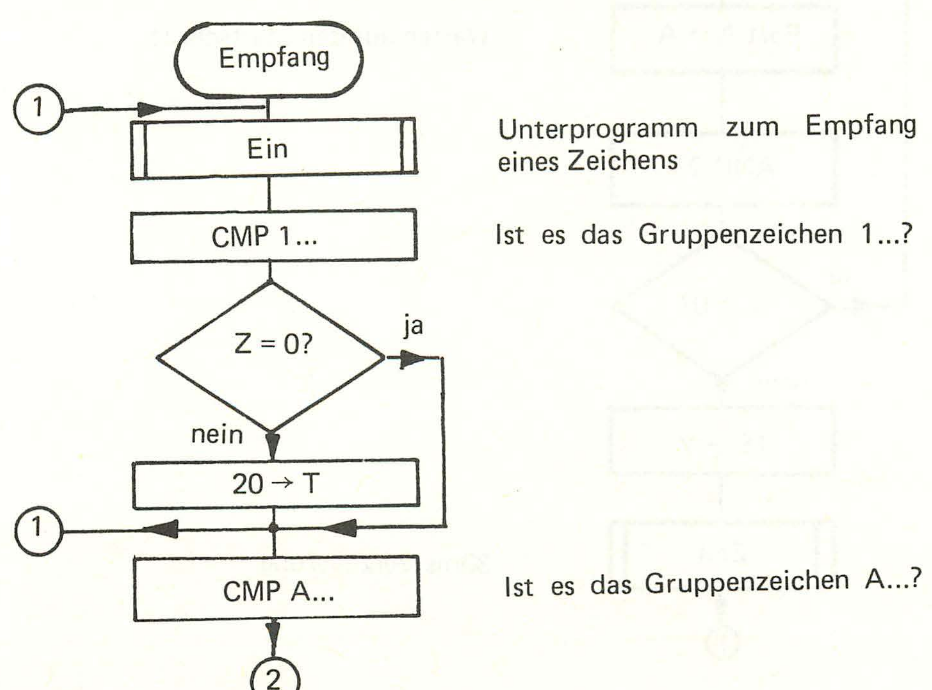
Dieses Programm soll die Impulsfolge, die am Empfangskanal anliegt, als 5 bit Zeichen aufnehmen. Wenn Gruppensteuerzeichen ankommen, soll es sich die neue Gruppenzugehörigkeit merken. Der 5 bit Code des Zeichens soll in den 6 bit Code umgewandelt werden.

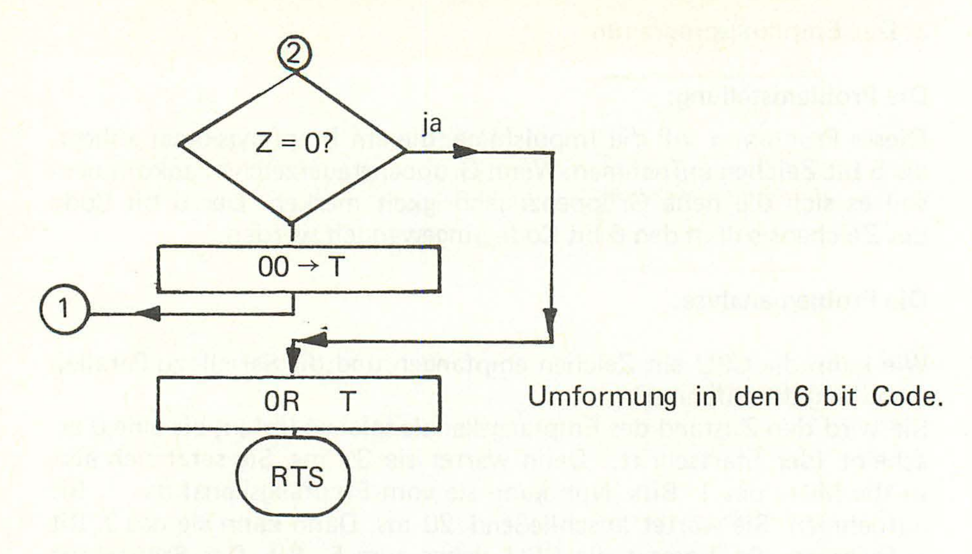
Die Problemanalyse:

Wie kann die CPU ein Zeichen empfangen und die Seriell- zu Parallelwandlung durchführen?
Sie wird den Zustand des Empfangskanals solange testen, bis eine 0 erscheint (der Startschritt). Dann wartet sie 30 ms. Sie setzt sich also in die Mitte des 1. Bits. Nun kann sie vom Empfangskanal das 1. Bit aufnehmen. Sie wartet anschließend 20 ms. Dann kann sie das 2. Bit aufnehmen. So kommt die CPU sicher zum 5. Bit. Der Stopschritt ist für die CPU nicht interessant.

Die Problemlösung:

Das Programm wird natürlich wieder als Unterprogramm formuliert.





Das Unterprogramm Ein:

Ein

Port A → A

Warten auf den Startschritt

AND 20

ja

Z = 0?

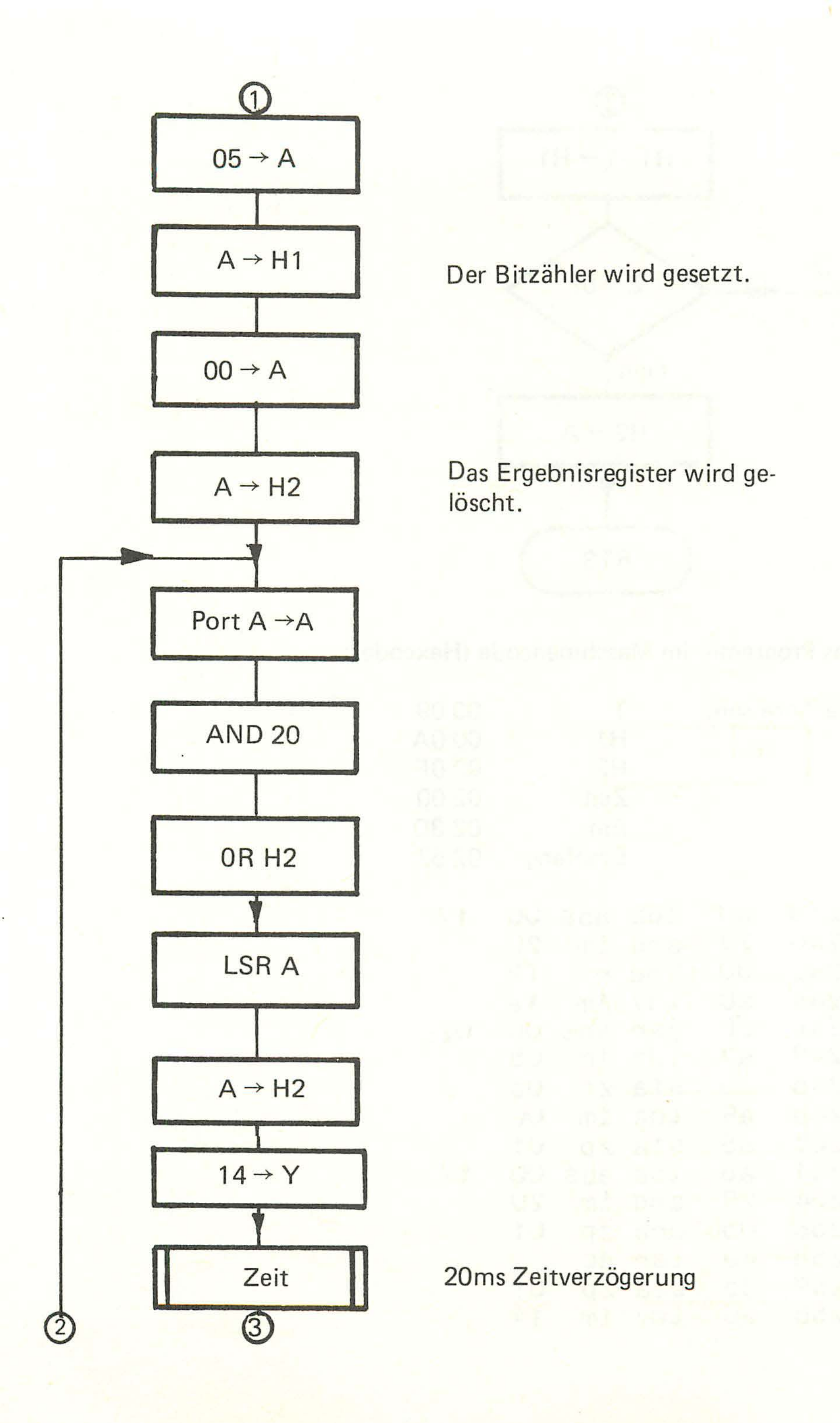
nein

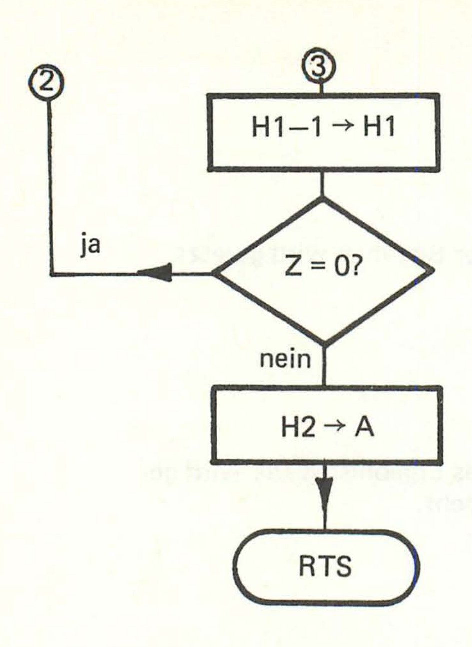
1E → Y

Zeit

30ms Verzögerung

1


1
05 → A
A → H1
Der Bitzähler wird gesetzt.
00 → A
A → H2
Das Ergebnisregister wird gelöscht.
Port A →A
AND 20
OR H2
LSR A
A → H2
14 → Y
Zeit
20ms Zeitverzögerung
2
3



Das Programm im Maschinencode (Hexcode):

Die Adressen:		
	T	00 09
	H1	00 0A
	H2	00 0F
	Zeit	02 00
	Ein	02 3D
	Empfang	02 67

```
023d  ad  lda abs 00  17
0240  29  and im  20
0242  d0  bne r   f9
0244  a0  ldy im  1e
0246  20  jsr abs 00  02
0249  a9  lda im  05
024b  85  sta zp  0a
024d  a9  lda im  00
024f  85  sta zp  0f
0251  ad  lda abs 00  17
0254  29  and im  20
0256  05  ora zp  0f
0258  4a  lsr ac
0259  85  sta zp  0f
025b  a0  ldy im  14
```

```
025d  20  jsr abs 00  02
0260  c6  dec zp  0a
0262  d0  bne r   ed
0264  a5  lda zp  0f
0266  60  rts imp
0267  20  jsr abs 3d  02
026a  c9  cmp im  09
026c  f0  beq r   f9
026e  c9  cmp im  1b
0270  d0  bne r   07
0272  a9  lda im  20
0274  85  sta zp  09
0276  4c  jmp abs 67  02
0279  c9  cmp im  1f
027b  d0  bne r   07
027d  a9  lda im  00
027f  85  sta zp  09
0281  4c  jmp abs 67  02
0284  05  ora zp  09
0286  60  rts imp
```

Der Speicherauszug:

```
026720 09 02 c9 02 f0 f9 c9 1b d0 07 a9 20 85 09 4c
027767 02 c9 1f d0 07 a9 00 85 09 4c 67 02 05 09 60
```

4. Einfache Testprogramme

Natürlich wollen wir unsere Programme testen. Die Funktionstüchtigkeit des Sendeprogramms können wir leicht mit dem folgenden Hauptprogramm prüfen.

Wurde der Fernschreiber eingeschaltet, so muß er erst in einen definierten Anfangszustand gebracht werden. Dies wird durch die Vorbereitung ausgeführt.

Vorbereitung:

```
LDA   IM    01       Der Ausgangsport wird de-
STA   AB    PADD     finiert
LDA   IM    00
STA   ZP    T        T wird gesetzt.
LDA   IM    A...     und der Fernschreiber auf
JSR   AB    Senden   Buchstaben geschaltet
LDA   IM    CR       Lade Wagenrücklauf
JSR   AB    Senden
LDA   IM    LF       Lade Zeilenvorschub.
JSR   AB    Senden
```

Nun kann ein beliebiges Programm folgen. Zum Beispiel das Drucken eines Buchstaben.

```
LDA   IM    Buchstabe
JSR   AB    Senden
BRK   IMP
```

Reagiert der Drucker auf Anhieb nicht einwandfrei, muß man mit Hilfe eines Oszillographen die Impulsmuster nachmessen. Meist stimmen die Baudraten nicht ganz überein. Dann sollten die Zeitkonstanten variiert werden.

Läuft das Sendeprogramm einwandfrei, können wir das Empfangsprogramm testen.
Die Vorbereitung bleibt gleich. Es folgt dann ein Unterprogrammsprung in das Unterprogramm Empfang. Das empfangene Zeichen soll wieder gesendet werden.

```
M     JSR   AB    Empfang
      JSR   AB    Senden
      JMP   AB    M
```

Wenn das Programm Empfang nicht einwandfrei laufen sollte, muß man auch hier die Konstanten der Zeitschleifen variieren, um die Baudrate genau einstellen zu können.

Wir wollen jetzt ein einfaches Programm entwickeln, das eine Vorbereitung für das nächste Programmbeispiel sein soll.
In ein Feld F laden wir unseren Namen, natürlich im 6 bit Code verschlüsselt. Die letzten Zeichen des Feldes sollen CR und LF sein. F sei die Anfangsadresse des Feldes und L dessen Länge (einschließlich der Sonderzeichen CR und LF) um eine vermehrt.

```
      START
      Vorbereitung
M1    LDX   IM    00
M2    STX   ZP    Z        Z ist ein Hilfsspeicher
      LDA   ABX   F
      JSR   AB    Senden
      LDX   ZP    Z
      INX   IMP
      CPX   IM    L
      BNE   R     M2
      BEQ   R     M1
```

Dieses Programm druckt den Inhalt des Feldes (unseren Namen) aus und beginnt dann in einer neuen Zeile wieder mit dem Ausdruck des Feldinhaltes.

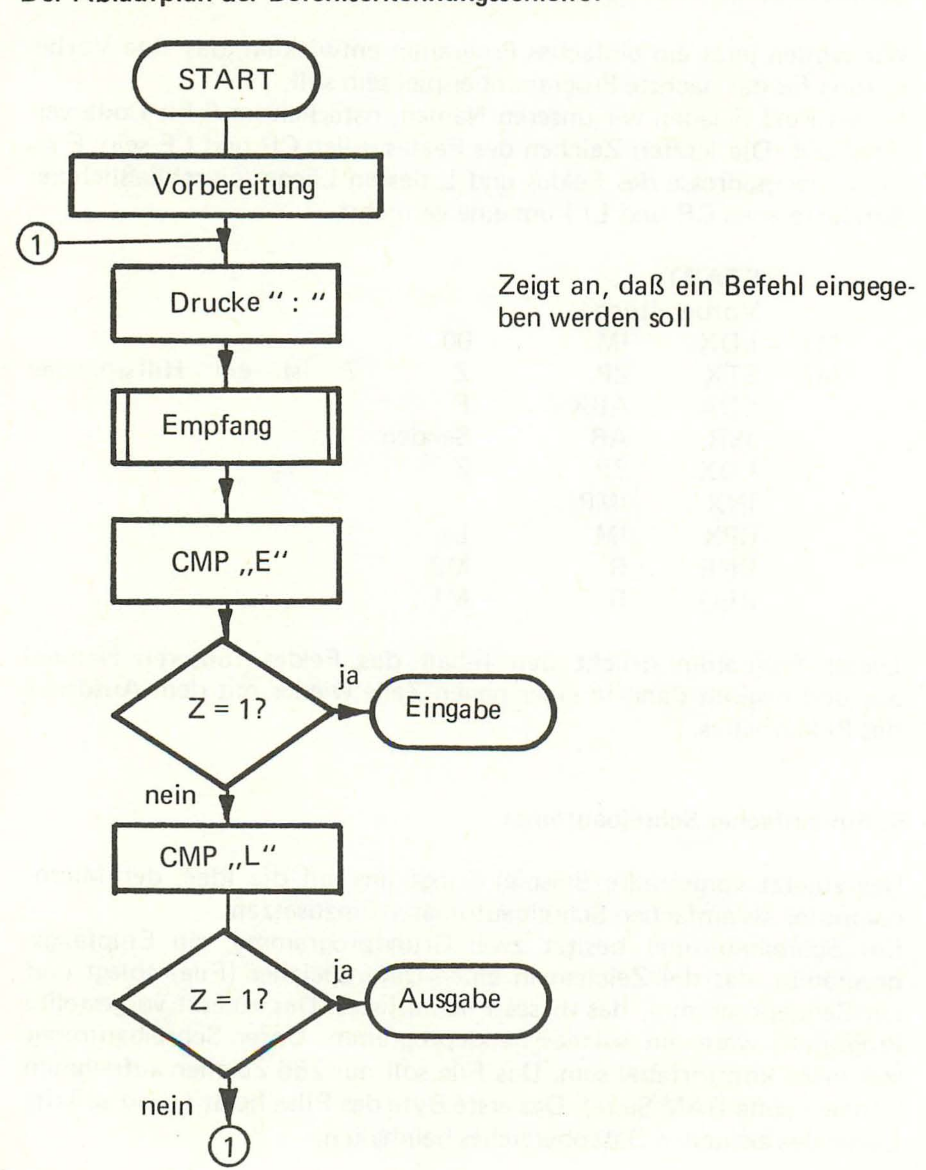
5. Ein einfacher Schreibautomat

Das zuletzt vorgestellte Beispiel bringt uns auf die Idee, den Microcomputer als einfachen Schreibautomaten einzusetzen.
Ein Schreibautomat besitzt zwei Grundprogramme, ein Empfangsprogramm, das die Zeichen in einen Datenspeicher (File) ablegt und ein Sendeprogramm, das dieses File auslistet. Das zuletzt vorgestellte Programm wäre ein solches Sendeprogramm. Unser Schreibautomat soll nicht komfortabel sein. Das File soll nur 256 Zeichen aufnehmen können (eine RAM-Seite). Das erste Byte des Files heißt L und soll die Länge des aktuellen Datenbereiches beinhalten.

Die Anfangsadresse des Files heißt F.
Unser Programm soll drei Befehle erkennen können.

1. E, dann beginnt das Einschreibeprogramm.
2. „Wer da?", dann wird das Einschreibeprogramm beendet.
3. L, dann wird das File ausgedruckt.

Der Ablaufplan der Befehlserkennungsschleife:

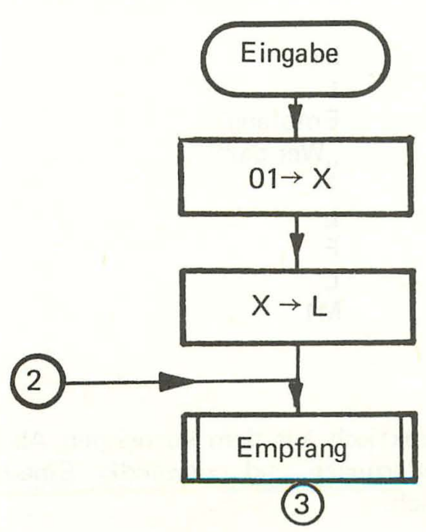


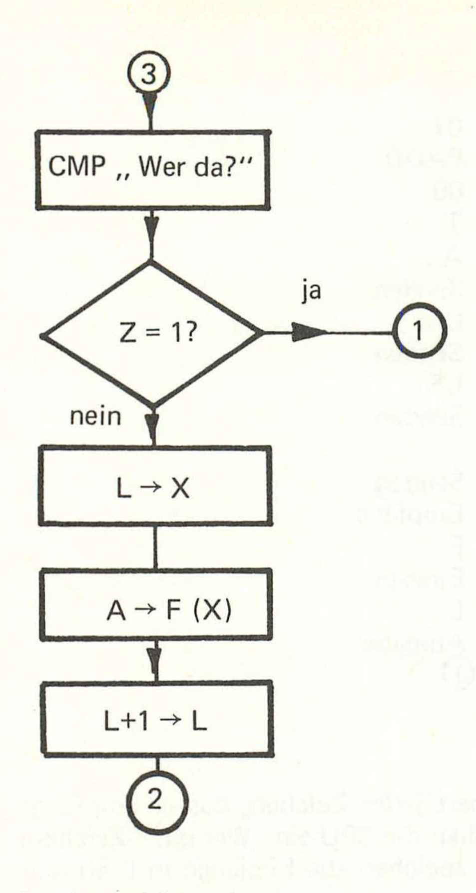
Im Assemblercode:

```
        LDA   IM   01
        STA   AB   PADD
        LDA   IM   00
        STA   ZP   T
        LDA   IM   A...
        JSR   AB   Senden
   ①    LDA   IM   CR
        JSR   AB   Senden
        LDA   IM   LF
        JSR   AB   Senden
        LDA   IM   :
        JSR   AB   Senden
        JSR   AB   Empfang
        CMP   IM   E
        BEQ   R    Eingabe
        CMP   IM   L
        BEQ   R    Ausgabe
        BNE   R    ①
```

Die Eingabe:

Der Programmteil Eingabe speichert jedes Zeichen, das er empfängt der Reihe nach im File ab. Empfängt die CPU ein „Wer da?"-Zeichen, bricht sie die Eingabeschleife ab, speichert die Filelänge in L ab und springt nach 1 zurück.





Im Assemblercode:

Eingabe	LDX	IM	01
	STX	AB	L
M1	JSR	AB	Empfang
	CMP	IM	„Wer da?"
	BEQ	R	1
	LDX	AB	L
	STA	ABX	F
	INC	AB	L
	JMP	AB	M1

Die Ausgabe:

Diesen Programmteil können wir praktisch aus dem vorherigen Abschnitt abschreiben. Nur kleine Änderungen sind notwendig. Einen Hilfsspeicher Z benötigen wir zusätzlich.

Der Drucker soll immer mit einer neuen Zeile beginnen.

```
Ausgabe  LDA   IM    CR
         JSR   AB    Senden
         LDA   IM    LF
         JSR   AB    Senden
         LDX   IM    01
M2       STX   ZP    Z
         LDA   ABX   F
         JSR   AB    Senden
         LDX   ZP    Z
         INX   IMP
         CPX   AB    L
         BNE   R     M2
         JMP   AB    (1)
```

6. Auslisten des Speicherinhaltes

In erster Linie wird der Anwender den Fernschreiber zur Erstellung von Programmdokumentationen heranziehen. Es ist sehr nützlich, wenn man eine Hardcopy des Programms erstellen kann.

Wir wollen nun ein Programm entwickeln, das einen Speicherauszug herstellt. Solche Speicherauszüge werden üblicherweise im Hexcode erstellt.

Als erstes sei das Programm Byte vorgestellt, das ein Byte im Hexcode ausdruckt.

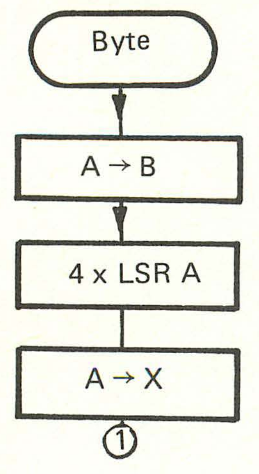
Das Unterprogramm Byte:

Die Problemstellung:

Das Unterprogramm soll ein Byte, das sich im Akkumulator befindet, im Hexcode ausgeben.

Die Problemlösung:

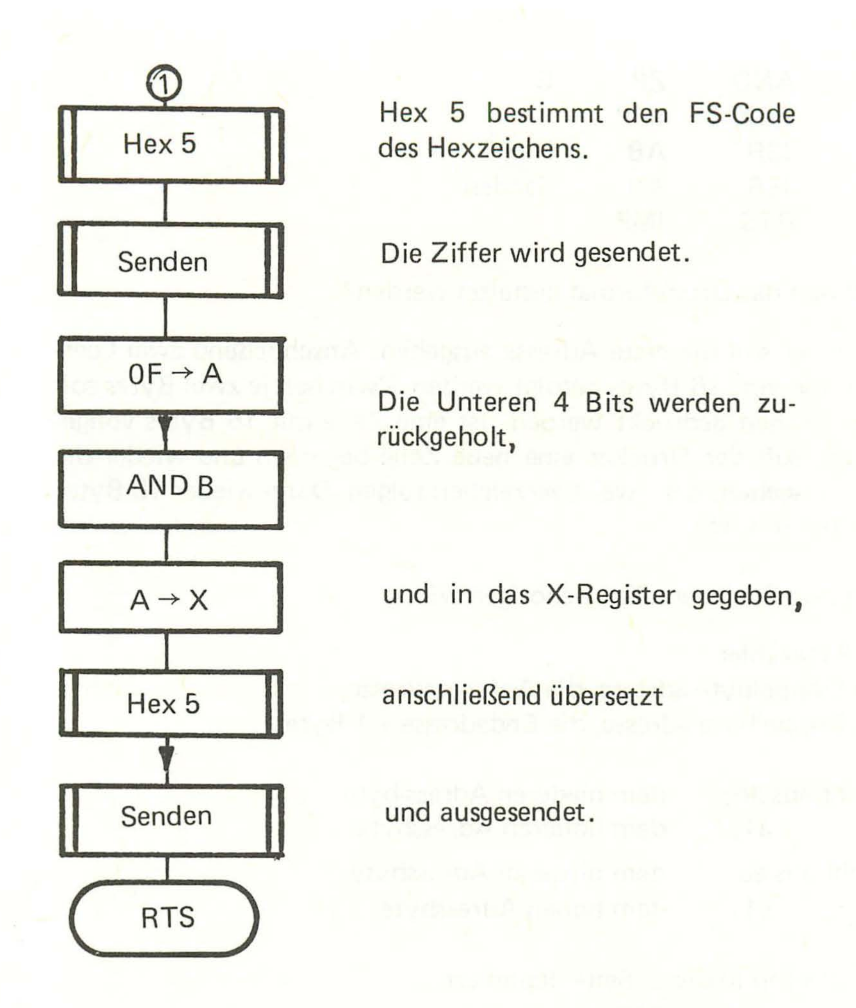
Das Byte wird in zwei 4 bit Hexzahlen aufgeteilt. Diese Hexzahlen übersetzt das Unterprogramm Hex 5, das wir in Kapitel 4.1.2 (2. Beispiel) kennenlernten, in den 6 bit Code. Anschließend sendet das Unterprogramm Senden diese Zeichen an den Drucker.



Das Byte wird in einem Hilfsspeicher B abgelegt.

Die oberen 4 Bits werden in eine Hexzahl umgewandelt,

und in das X-Register transportiert.



Byte im Assemblercode:

```
STA    ZP    B
LSR    AC
LSR    AC
LSR    AC
LSR    AC
TAX    IMP
JSR    AB    Hex 5
JSR    AB    Senden
LDA    IM    0F
```

AND	ZP	B
TAX	IMP	
JSR	AB	Hex 5
JSR	AB	Senden
RTS	IMP	

Wie soll nun das Druckformat gestaltet werden?

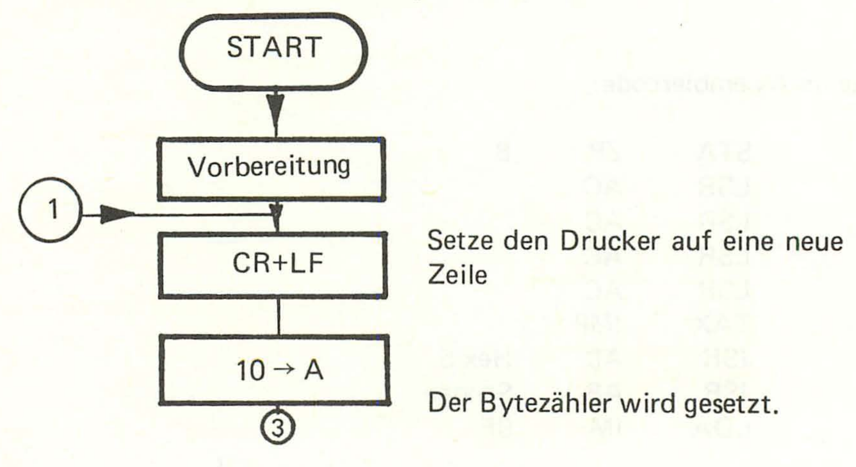
Der Drucker soll die erste Adresse ausgeben. Anschließend zwei Leerzeichen, die von 16 Bytes gefolgt werden. Zwischen je zwei Bytes soll ein Leerzeichen gedruckt werden. Ist eine Zeile mit 16 Bytes vollgeschrieben, soll der Drucker eine neue Zeile beginnen und wieder die Adresse ausgeben, der zwei Leerzeichen folgen. Dann wieder 16 Bytes ausdrucken u. s. w..

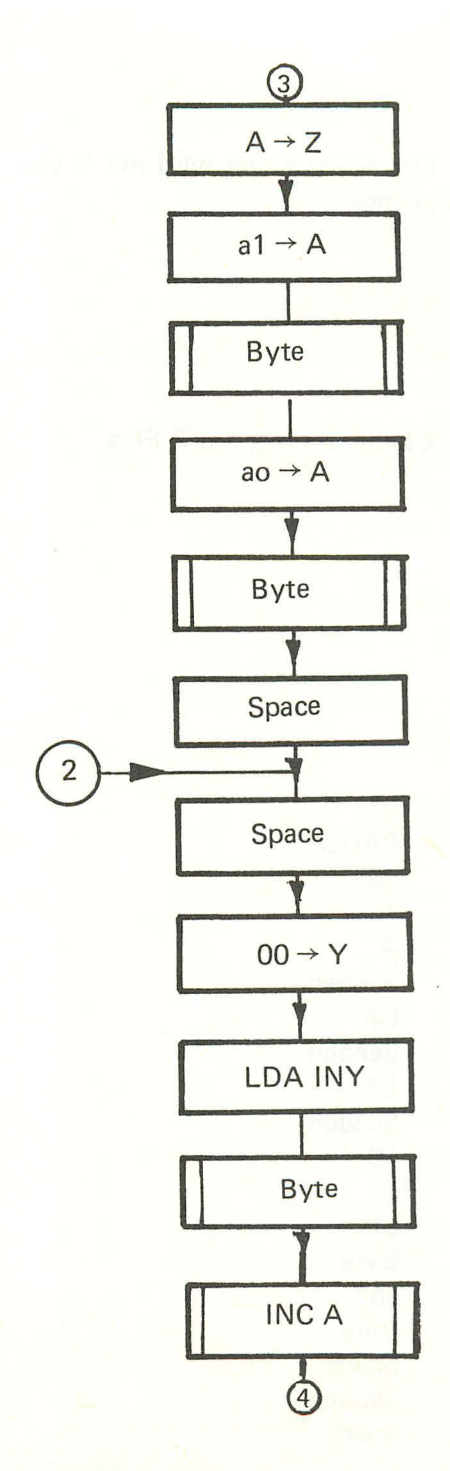
An weiteren Speicherzellen benötigen wir:

Z, den Bytezähler,
A, eine Doppelbyte adresse, die Anfangsadresse,
E, eine Doppelbyte adresse, die Endadresse + 1 Byte.

A besteht aus	ao,	dem niederen Adress-byte
	a1,	dem höheren Adressbyte.
E besteht aus	eo,	dem niederen Adressbyte,
	e1,	dem hohen Adressbyte.

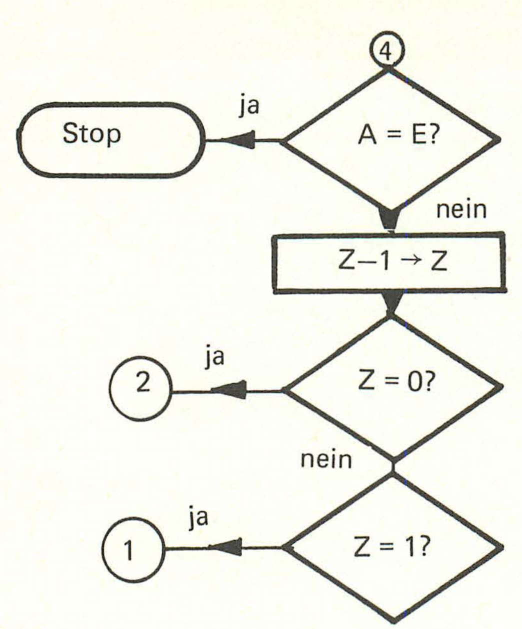
Z bis E werden in die 0. Seite deponiert.





Das Datenbyte wird mittels der indirekten durch Y indizierten Adressierung in A geholt und ausgedruckt.

Der Adresszeiger wird incrementiert.



Der Adresszeiger wird mit E verglichen.

Z bedeutet hier das Z-Flag.

Das Hauptprogramm im Assemblercode:

Vorbereitung:	LDA	IM	01
	STA	AB	PADD
	LDA	IM	00
	STA	ZP	T
	LDA	IM	A...
	JSR	AB	Senden
CR+LF ①	LDA	IM	CR
	JSR	AB	Senden
	LDA	IM	LF
	JSR	AB	Senden
	LDA	IM	10
	STA	ZP	Z
	LDA	ZP	a1
	JSR	AB	Byte
	LDA	ZP	ao
	JSR	AB	Byte
2x Space	LDA	IM	Space
	JSR	AB	Senden
②	LDA	IM	Space

```
INC A                 JSR   AB    Senden
                      LDY   IM    00
                      LDA   INY   ao
                      JSR   AB    Byte
                      INC   ZP    ao
                      BCC   R     M1
                      INC   ZP    a1
              M1      LDA   ZP    a1
                      CMP   ZP    e1
                      BNE   R     M2
                      LDA   ZP    ao
                      CMP   ZP    eo
                      BNE   R     M2
                      BRK   IMP          Hier kann man
                      NOP   IMP          auch einen
                      NOP   IMP          Sprungbefehl ein-
              M2      DEC   ZP    Z      fügen.
                      BNE   R     ②
                      BEQ   R     ①
```

Bei der Übertragung des Programms in den Maschinencode muß auf die folgenden Punkte geachtet werden.

1. Es kann leicht geschehen, daß eine Datenzelle in der 0. Seite zweifach genutzt wird, was zu ganz unerklärlichen Fehlern führen kann. Daher: Unbedingt genau die Verwendungen der Datenzellen beachten.

2. Das Programm Hex 5 aus Kapitel 4.1.3 (2. Beispiel) kann nicht einfach übernommen werden, da der RAM-Bereich ab 0200 durch das Sendeprogramm belegt ist. Man muß Hex 5 also in einen anderen Speicherbereich ablegen.



NOTIZEN



Elektronik Schaltungen,
4. völlig neu überarbeitete Auflage, W. Hofacker

Die ideale Schaltungssammlung zum Basteln und Experimentieren. Schaltungen mit Operationsverstärkern, Spannungsreglern, TTL, C-MOS Schaltkreisen. MOS Uhr mit Wecker, elektronischer Würfel, Musik Synthisizer, Timer 555 Anwendungen, Experimentieranleitungen und viele andere hochinteressante Schaltbeispiele tlw. mit Printvorlage. 64 Seiten Inhalt.

DM 5,–



IC - Bauanleitungs -Handbuch -IC -KIT , C. Lorenz

Ein Bauanleitungsbuch mit vielen hochinteressanten Bauanleitungen aus dem Bereich der LSI Schaltungstechnik. Schaltbeispiele mit Printvorlagen zum Selbstherstellen der Leiterplatten mit genauesten Beschreibungen. Hochaktuell und brandneu: Funktionsgenerator XR 2206, MOS-Uhr mit Wecker, Schlummerautomatik und programmierbarem Wecktongenerator, Sensortastenwahl, IC-Netzteil, Funktionsgenerator 8038 neuartige Transistorzündung, 35 W NF-Verstärker, Experimentieranleitung und Grundkurs über Flip Flops, Experimente mit Digitalschaltungen u. v. a. Zu allen Schaltungen finden Sie Platinvorlagen oder Sie können die Experimentierschaltungen auf der Experimentierplatine WH-1 g durchführen. Über 125 Seiten.

DM 19,80

41

Experimentierplatine mit Sockel, Stecker und Füßen Typ WH-1g für 40,28,24,16 und 14 polige DIL -Gehäuse

DM 79,–



Feldeffekttransistoren, C. Lorenz

Der Feldeffekttransistor (FET) gehört heute zu den interessantesten Bauteilen überhaupt. Wie man damit experimentiert, wie man seine Funktion versteht und wie man damit brauchbare und hochinteressante Schaltungen aufbauen kann, zeigt Ihnen dieses Buch. Grundlagen, Kennlinienfelder, Tabellen, Berechnungsgrundlagen, Rechenbeispiele, Anschlußbilder und eine Vergleichsliste für Feldeffekttransistoren bilden den Kern dieser umfangreichen Darstellung. Alles in allem finden Sie hier eine praxisnahe und komplette Arbeitsunterlage, mit der Sie im Beruf und auch im Hobby erfolgreich arbeiten können. Über 45 Seiten.

DM 5,–



Elektronik und Radio, C. Lorenz
4. Auflage. Völlig neu bearbeitet und stark erweitert.

Eine Einführung in die Radiotechnik, wie man sie nicht alle Tage findet. Eine sehr geschickt gemachte Einführung mit vielen Schaltungen, Bauanleitungen und genauesten Funktionsbeschreibungen. Vom einfachen Diodenempfänger (Detektor) bis zu interessanten Sender- und Empfängerschaltungen. (Minispione) Viele hundert Bilder zeigen Ihnen genau, wie Sie beim Experimentieren vorgehen müssen. IC-Radio, IC-Sender, Antennen, Berechnungsgrundlagen, Tabellen u. v. a. Über 150 Seiten **DM 19,80**



IC -Niederfrequenzverstärker , C. Lorenz

Grundlagen der integrierten NF-Verstärker, Berechnung von kompletten IC-NF-Verstärkerstufen. Anwendungsbeispiele mit den interessantesten und gebräuchlichsten Standard IC-NF-Verstärkern wie TBA 800, TBA 830, usw. Printvorlagen, Auswahltabellen, Experimentieranleitungen und Anschlußbilder machen dieses Buch zu einem unentbehrlichen Begleiter für alle, die sich mit NF-Verstärkern beschäftigen wollen. Über 65 Seiten.

DM 9,80

12

BIS BUCH, Beispiele integrierter Schaltungen, H. Bernstein

Auf über 130 Seiten Anwendungsbeispiele mit integrierten Schaltkreisen. Zeitgeber 555, Funktionsgenerator ICL 8038, Opto Elektronik, Operationsverstärker, Analogschalter, Digital-Analog-Wandler, Analoge Rechenbausteine, Schreib-Lese-Speicher (RAM), Festwertspeicher (ROM), Speicherschaltungen, Uhrenbausteine u. v. a. **DM 19,80**

13

HEH, Hobby Elektronik Handbuch, C. Lorenz

Das Schaltungsbuch für jeden Hobbyelektroniker. Schaltbeispiele und Bauanleitungen aus dem gesamten Hobbybereich. Lichtorgeln, Eiswarngerät fürs Auto, Alarmanlagen, Metallsuchgerät, PLL-Schaltungen, Logik-Tester, Funktionsgeneratoren u. v. a. Über 55 Seiten. **DM 9,80**

14

IC - Vergleichsliste, C. Lorenz

Vergleichsliste für digitale und lineare integrierte Schaltkreise.

Standard TTL, Low Power Schottky TTL, C MOS, Triacs Thyristoren, Optoelektronik, Operationsverstärker, Spannungskomparatoren, Spannungsregler, NF-Verstärker u. v. a. Funktionsvergleichsliste CMOS zu TTL. Vergleichstabelle für Transistoren und Dioden sowie Darlingtontransistoren. Eine Vergleichsliste, die man immer wieder braucht.
DM 29,80

15

Opto -Handbuch, Handbuch für Optoelektronik, C. Lorenz

Das Handbuch für die gesamte Optoelektronik. Eine Einführung und ein ideales Nachschlagwerk. Grundlagen, Definitionen aller Kenngrößen, Opto-Lexikon, Berechnungsgrundlagen, Rechenbeispiele, Schaltbeispiele: Lichtsender, Lichtempfänger, Anzeigen, Infrarot Detektoren, Lichtmeßgerät, Optokoppler, Pegelschalter, Opto-Vergleichsliste. Anschlußbilder wichtiger 7-Segment-Anzeigen u. v. a. Über 106 Seiten. **DM 19,80**

16

C MOS Einführung, Entwurf, Schaltbeispiele, Teil 1, H. Bernstein

Vom C MOS Gatterbaustein über Schieberegister und Zähler bis hin zum C MOS Schreib-Lesespeicher. Insgesamt werden neunzehn interessante und bekannte C MOS Schaltkreise beschrieben. Zu jedem Bauelement sind genaue Daten, Schaltbild und Anwendungsbeispiele angegeben. Im großen Applikationsteil finden Sie: C MOS-Kippstufen, Addierwerke und Rechenschaltungen, Digital Analog Wandler, Schieberegister für analoge Spannungen, Multiplexsysteme für analoge Signale u. v. a. Eine komplette Einführung und gut geeignet für das Selbststudium der C MOS Technik. Über 140 Seiten. **DM 19,80**

17

C MOS Entwurf und Schaltbeispiele, Teil 2, H. Bernstein

Fortsetzung von Teil 1. Anwendungsbeispiele mit genauen Schaltungsbeschreibungen und Bauelementeunterlagen. Daten, Anschlußbelegungen weiterer wichtiger hochintegrierter C MOS Elemente. Ein komplettes Arbeits- und Experimentierbuch. C-MOS Uhrenschaltungen, Schieberegisterschaltungen, Parallel-Serien Umsetzung, statische und dynamische Speicherschaltungen, Zählschaltungen, Digital Analog Wandler, Analog Digital Wandler, Digital Voltmeter, I/O Registerschaltungen, Codier und Dekodierschaltungen. RAM und ROM Anwendungen. Über 140 Seiten. **DM 19,80**

18

C MOS Entwurf und Schaltbeispiele, Teil 3, H. Bernstein

Fortsetzung von Teil 2. Eine sehr umfangreiche Applikationssammlung mit hochintegrierten C MOS Elementen. Rechnerschaltungen, Speicher- und Steuerschaltungen, Multiplex- und Datenbussysteme, Uhrenschaltungen, PLL-Schaltungen, Liquid Cristal Anzeigen und deren Treiberschaltungen, Optoelektronik in Verbindung mit C MOS. Grundlagen, Aufbau und Wirkungsweise der Prozeßrechentechnik, Arithmetische Logische Einheiten (ALU) und andere wichtige Funktionen aus der Prozeßrechentechnik. RAMs, ROMs und FIFO-Speicherschaltungen. Über 140 Seiten. **DM 19,80**

19

IC Experimentier Handbuch - IC -EX, C. Lorenz C. Lorenz

Eine sehr umfangreiche Schaltungssammlung und Bauanleitunssammlung mit neuesten integrierten Bausteinen. Neue, jedoch beim Fachhandel erhältliche Standard ICs. Rechnerschaltungen, Mikroprozessoren, I/O Schaltungen, druckende und anzeigende Rechner, Stoppuhren, Zählerschaltungen, Digitalvoltmeter, professioneller Synthisizer, Hilfsschaltungen für den Elektronik Experimenter, Analog Digital Wandler, Frequenzzähler u. v. a. hochinteressante Bauanleitungen. Viele Schaltungen können auf der IC KIT Experimentierplatine WH–1g aufgebaut werden. Über 120 Seiten. **DM 19,80**



Operationsverstärker, Grundlagen und Schaltbeispiele, C. Lorenz

Dieses Buch umfaßt das gesamte Gebiet der linearen Schaltungstechnik und stellt ein in dieser Preislage bisher noch nie dagewesen Nachschlagwerk und Einführungshandbuch dar. Bestens geeignet für das Selbststudium. Nach einer pädagogisch geschickt gemachten Einführung folgen theoretische Arbeitsunterlagen und die zugehörigen Schaltbeispiele mit Daten und Gehäuseanschlüssen. Dieses wertvolle Buch dürfte seinen Platz auch bei Ihren Arbeitsunterlagen finden, und wird dann immer von Nutzen sein, wenn es um die Lösung von nicht routinemäßigen Aufgaben geht. **DM 19,80**



Digitaltechnik Grundkurs (TTL -C MOS - MOS und Software), C. Lorenz

Ein Einführungskurs in die Digitaltechnik für Anfänger und Fortgeschrittene. Ein Fachbuch für den programmierten Selbstunterricht. Der ideale Kurzlehrgang für das Selbststudium. Der Kurs vermittelt Ihnen alle wichtigen Grundkenntnisse vom TTL-Gatter bis zum Mikroprozessor und Lösung von Schaltungsaufgaben durch Software. Viele Versuchsaufbauten und Experimente aus diesem Kurs können auf der IC-KIT Platine WH–1g durchgeführt werden. Grundlagen, Gatter, Zähler, programmierbare Zähler, IC-Tester, Schieberegister, Speicher, Mikroprozessoren u. v. a. Über 130 Seiten. **DM 19,80**



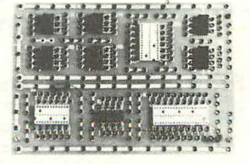


Experimentierplatine WH-1g dazu, Best.Nr. 41 — DM 79,–



Experimentierplatine WH - 1 g
Abmessungen: 210 mm x 150 mm

Experimentierplatine WH - 1 g
fertig aufgebaut und mit Sockeln bestückt.





Mikroprozessoren, Eigenschaften und Aufbau Teil 1 , H. Bernstein

Grundlagen, Eigenschaften und Aufbau von Mikroprozessoren. Organisation von Recheneinheiten und Mikroprogrammen. Programmierung und Klassifizierung von Mikroprozessoren. Ablaufdiagramm, Flußdiagramm. Ein Chip-Technik und Multi Chip-Technik, Transfer- und Sprungfunktionen. Speichertechnik: RAMs, ROMs, FIFO, FILO. Programmierbare logische Arrays (PLA)
Anwendungsbeispiele und Anwendungsbereiche. Über 120 Seiten. **DM 19,80**



Elektronik Grundkurs (Kurzlehrgang Elektronik), C. Lorenz

Eine leichtverständliche und pädagogisch geschickt gemachte Einführung in die Technik der elektronischen Schaltungen. Ein Kurzlehrgang und Schnellkurs zugleich. Aber auch ein recht brauchbares Nachschlagewerk für den fortgeschrittenen Elektroniker. Mit wenig Mühe können Sie sich hier die Grundkenntnisse der elektronischen Schaltungspraxis aneignen. Das Buch schafft die Voraussetzungen für ein erfolgreiches und sicheres Arbeiten mit interessanten Schaltkreisen modernster Technologien. Unentbehrlich für das Experimentieren mit den heutigen modernen hochintegrierten Schaltkreisen. Über 150 Seiten.
DM 9,80



Mikrocomputer Technik, Hans Peter Blomeyer-Bartenstein

(völlig neu überarbeitete Auflage, Herbst 1979)
In diesem Buche finden Sie eine umfassende, einführende und weiterführende Hilfe zum Einstieg in die Mikrocomputertechnik mit vielen Schalt- und Programmierbeispielen. Als praktische Betrachtungsgrundlage dient das supermoderne Microcomputerkonzept Z80A von ZILOG. Das Buch geht auf alle wichtigen Zusammenhänge ein und erklärt diese dem Leser so ausführlich, daß kaum noch Fragen offen bleiben. Über 240 Seiten **DM 29,80**



Hobby Computer Handbuch, C. Lorenz

Eine leicht verständliche Einführung in die Microcomputertechnik.

Diese sehr umfangreiche Einführung in die Microcomputertechnik dürfte zu diesem Preise einmalig sein. Auf über 450 Seiten finden Sie: Grundlagen der Computer- und Microcomputer-Technik, Was ist ein Microcomputer?, Microcomputer KITs, Einplatinencomputer, KIM, MIKIT, Z80 KIT, 6800 KIT, NEC 8080 KIT u. v. a. Genaue Beschreibungen der wichtigsten Mikroprozessortypen. Zusammenstellung und Beschreibung der modernen Personal Computer. (IMSAI, CROMEMCO, CAT, OSI, POLY 88 u. v. a.) Kompaktcomputer wie PET und TR-8. Interfactechniken, Ein/Ausgabegeräte, ROMs, RAMs, Programmiergeräte für PROMs. Löschgeräte für EPROMs u. v. a. mehr. Das ideale allumfassende Buch für den Microcomputertechniker. Für Industrieanwendung ebenso geeignet, wie für den Hobby-Computer Fan. Über 450 Seiten **DM 29,80**



Mikroprozessor Teil 2, H. Bernstein

Die Fortsetzung unseres ersten so erfolgreichen Buches über Mikroprozessoren. Technologie von Mikroprozessor- und Speicherbausteinen. Festwertspeicher, PROM, REPROM, FIFO, Schieberegister, MPR-Register, ARL-Register, SAR-Register. Aufbau eines Mikroprozessorsystems mit 8080, RAM- und ROM Schnittstellen. Befehlssatz 8080. Über 120 Seiten. **DM 19,80**



Mikroprozessor Software Handbuch MSH, C. Lorenz

Grundlagen und Einführung in die Mikroprogrammierung. Grundlagen und Einführung in die wichtigsten Programmiersprachen (BASIC, FORTRAN, ASSEMBLER-Sprachen) Zusammenstellung der wichtigsten Befehlslisten: 8080, Z80, M 6800, National, Fairchild, etc.
Ein Software Handbuch für jeden der mit Mikroprozessoren oder Mikrocomputern zu tun hat. Über 200 Seiten. **Preis DM 29,80**



Lexikon und Wörterbuch für Elektronik und Mikroprozessortechnik, LEM, C. Lorenz

Ein Hilfs- und Arbeitsbuch für jeden der sich heute mit der modernsten Elektronik beschäftigt. Viele engl. Ausdrücke werden heute in der Elektronik, Computer- und Mikroprozessortechnik verwendet und oft fehlt uns eine genaueste und präzise Erklärung. Dieses Buch übersetzt Ihnen den englischen Fachausdruck und gibt Ihnen zusätzlich noch eine deutsche Erläuterung und Erklärung dieses Begriffes und was es damit auf sich hat. Ein Lexikon und Wörterbuch in einem einzigen Buch vereinigt. Das Buch, das Sie schon lange gesucht haben. Ca. 250 Seiten. **Preis DM 29,80**



Mikrocomputer Datenbuch, C. Lorenz

Zusammenstellung der wichtigsten Mikroprozessordaten.

Eine übersichtliche und sehr informative Zusammenstellung der wichtigsten Mikroprozessorbausteine auf dem Markt. 8080A, 8085, 8048, Z80, Z8, 6500, 6800, 2650, 1802, F8, 3870, SC/MP, PACE u. v. a. Daten, Anschlußbilder, wichtige technische und elektrische Daten, Architektur, grundlegende Eigenschaften. Zu jedem Mikroprozessor werden dann auch noch die peripheren Bausteine sowie RAM und ROM Elemente behandelt. Das ideale Handbuch für jeden modernen Elektroniker.

DM 49,–



Aktivtraining - Mikrocomputer

Der ideale Einführungskurs in die Mikrocomputertechnik anhand des Mikroprozessors 8080 (8085). Viele interessante Anwendungsbeispiele. Ideal zum Selbststudium geeignet. Auch die wichtigsten peripheren Einheiten werden besprochen und aufgezeichnet. Erscheint Ende 1979/ Anfang 1980 als gebundenes Buch und nicht wie vorher angekündigt als Sammelordner. Umfang ungefähr 400 – 500 Seiten. **DM 49,–**



57 Programme in BASIC, C. Lorenz

Ein Buch mit technisch-wissenschaftlichen Programmen und einer großen Anzahl von Spielprogrammen in BASIC. (Games) Ein Buch für jeden, der sich mit dem faszinierenden Hobby der Mikrocomputertechnik befassen will. Alle Listings sind in BASIC und können auf den meisten Personal Computer Systemen gefahren werden.

DM 39,--



ATARI BASIC Handbuch

Das komplette Einführungs- und Programmierhandbuch für die neuen ATARI-Computer ATARI 400 und ATARI 800. Anhand von vielen Beispielen wird die Leistungsfähigkeit des ATARI Computer-Systems gezeigt. Auch für den Anfänger ideal als Einstieg gedacht. Die ATARI-Computer sind mit Shepardon's BASIC ausgerüstet. Erscheint voraussichtlich

Erscheint Ende 1980 **DM 29,80**

Mikrocomputer Programmierbeispiele für 2650, Dr. J. Hatzenbichler



Eine Einführung in die Programmierung von Mikrocomputern an Hand des Prozessors 2650 von Signetics. Viele Programmierbeispiele in Maschinensprache, die Sie auf einem preiswerten Mikroprozessorsystem MIKIT 2650-P2 ausführen können. Zeitschleifenprogramme, Blinkschaltung, Lauflicht, Stufenzähler, Elektronischer Würfel, Stoppuhr, Reaktionszeittester, Computer Musik Programm u. v. a. Jeder Befehl wird genau erläutert und an Hand eines ausführlichen Flußdiagrammes erklärt. Jedes Programm liegt in Form eines Computer-Listings vor. Jedes Programm ist genauestens beschrieben. Sie können so auf einfache Weise die Zusammenhänge erkennen und erwerben damit die Grundkenntnisse zur Erstellung Ihrer eigenen Programme. Sie erkennen, wie man mit nur einer Schaltung (Mikrocomputer) unendlich viele praktische Anwendungsschaltungen realisieren kann. Zu diesem Buche ist auch ein komplett aufgebautes und getestetes Mikrocomputersystem erhältlich, auf dem Sie alle beschriebenen Programme selbst ausführen können. Über 120 Seiten **DM 19,80**



TINY BASIC Handbuch

Eine Programmiersprache ist um so leistungsfähiger, je einfacher eine gewünschte Funktion dargestellt werden kann. TINY BASIC ist eine abgemagerte BASIC-Version, die mit ca. 2 K RAM auskommt. Dieses Buch zeigt Ihnen, wie Sie eine solche preiswerte BASIC-Version auf Ihrem KIM-1 oder einem anderen 6502-System implementieren können. Komplette Hard- und Softwarebeschreibungen, Programmiertricks, Programmierbeispiele und vieles andere mehr. **DM 19,80**

Der freundliche Computer

Ein Microcomputerbuch für jedermann. Dieses Buch sollte jeder lesen, der auch in Zukunft noch mitreden möchte. Was macht man mit einem Microcomputer? Die verschiedenen Computersprachen? Welche Computersprache ist die beste? Wie lerne ich eine Computersprache? Zusammenfassung verschiedener Computersprachen wie BASIC, MUMPS, PL/M, FOCAL, FORTRAN, ASSEMBLER. Vergleich dieser Computersprachen untereinander und Gegenüberstellung. Einkaufsführer für Microcomputer und Peripherie. Besprechung der einzelnen Teile des Microcomputers, ausführliches Stichwortverzeichnis der Fachausdrücke und v. a. m. **Erscheint Mitte 1981** **DM 29,80**



Microcomputer und Roboter

Ein Buch für denjenigen, der sich externe Schaltungen für Microcomputer bauen möchte, die roboterartige Funktionen ausführen können. Spracherkennung, Analog-/Digital-Wandler, Digital-/Analog-Wandler, Ultraschallsensoren, Lichtschranken, Tonerzeugung u. v. a.

Erscheint Ende 1981 **DM 29,80**



1000 Elektronik Schaltungen

Ein ideales Handbuch für jeden, der öfter eine Schaltung zur Lösung eines bestimmten Problems sucht. Tausend Schaltungen aus fast allen Bereichen der Elektronik. Industrielle Steuerschaltungen, Microcomputer, Peripherie, Hobby-Elektronik-Schaltungen u. v. a. mehr. Ein Buch, daß bei keinem Elektroniker fehlen sollte. Erscheint Mitte 1981 **DM 49,–**



Oszillographen Handbuch

Ein Buch für jeden, der seinen Oszillographen optimal nutzen will. Der Anfänger, der noch nie einen Oszillographen benutzt hat, wird auf einfache Weise mit der Technik und Handhabung vertraut gemacht. Auch die Anwendung in Zusammenhang mit den modernen Mikrocomputersystemen wird beschrieben. Oszillograph als alphanumerisches Darstellungsgerät u. v. mehr. Erscheint ca. Mitte 1981 **DM 19,80**



TTL - Experimentierbuch

Eine kleine Einführung in die Digitaltechnik

Grundlagen der Digitaltechnik kurz erklärt. Bauanleitung für einen praktischen Logik-Tester. Viele Experimente und Anwendungsschaltungen mit dem TTL Gatterbaustein 7400. Das ideale Einführungsbuch in die Digitaltechnik. DM 5,–



CMOS - Experimentierbuch

Eine kleine Einführung in die CMOS Schaltungstechnik

CMOS Grundlagen, Behandlungshinweise für CMOS Bausteine, Zusammenstellung der wichtigsten CMOS Bausteine und deren Anschlußbilder. Viele praktische CMOS Schaltbeispiele. Ideal für jeden Elektroniker. DM 5,–



Praktische Antennentechnik, C. Lorenz

Ein Buch für jeden Funkamateur oder Hobbyfunker. Grundlagen, Einführung, praktische Beispiele, Berechnungsgrundlagen u. v. a. mehr. Erscheint Ende 1981 **DM 19,80**



Handbuch für SC/MP – Das Handbuch für INS8060 und INS8070, C. Lorenz

Eine Anleitung zum Aufbau eines eigenen Microcomputersystems. Komplett mit allen Unterlagen (Schaltbilder, Printvorlagen, Anleitungen). Systemdaten: 4/8K RAM, residentes BASIC, residenter Assembler/Editor/Debugger, TV-Interface, Floppy Disk Interface, Cassetten-Interface. **DM 29,80**



6502 Microcomputer Programmierung, P. Heuer,

Ein deutsches Anleitungsbuch zum Einstieg in die Microcomputertechnik. Als System kann ein KIM-1 zur Ausführung der Programmierbeispiele verwendet werden. Die Maschinenprogramme lassen sich jedoch auch auf jedes 6502 System umschreiben. Daher ist dieses Buch auch für alle anderen 6502-System-Besitzer interessant. (AIM, SYM, CBM, Challenger, APPLE, ATARI, PET und PC100 sowie PC1000. **DM 29,80**



Programmierhandbuch für PET, C. Lorenz

Ein Handbuch für jeden PET-Besitzer und solche die es werden wollen. Grundlagen, Einführung und viele Programmierbeispiele. Einführung in die Maschinensprache, Ein-/Ausgabeprogrammierung, Spracheingabe für PET, Graphik mit dem PET, Computermusik, Programmierung des IEC-Ports, viele Programmbeschreibungen, Datenblätter, Programmbeschreibungen von interessanten Geschäftsprogrammen. 325 Seiten **DM 29,80**

111

Programmieren mit TRS-80, M. Stübs

Ausführliche Beschreibung des TRS-80 Computers von Radio Shack. Betriebssystem, Level II BASIC, Graphik, Mathematik und Logik, Programmiertricks, TRS-80 Floppy, Verbindung zur Außenwelt, RAM-Erweiterung u. v. a. mehr. Viele Programmierbeispiele mit ausführlicher Beschreibung runden das Buch ab und machen es zu einer wichtigen Informationsquelle für jeden, der vor der Anschaffung eines TRS-80 steht. Dem TRS-80-Besitzer wird es stets ein wertvoller Begleiter sein. **DM 29,80**

112

PASCAL Programmierhandbuch, C. Lorenz

Eine leicht verständliche Einführung in die PASCAL-Programmiersprache mit vielen Beispielen. Auch auf eine TINY-PASCAL-Version wird eingegangen. Viele Programmierbeispiele. **DM 29,80**

113

BASIC Programmierhandbuch, C. Lorenz

Einführung und Nachschlagewerk. Speziell für die BASIC-Versionen der modernen Microcomputersysteme. Jeder Befehl wird ausführlich beschrieben und ein Beispielprogramm gezeigt. Sehr übersichtlich und praktisch. Am Schluß finden Sie ein komplettes BASIC-Programm, das Ihnen über einen Computer BASIC lehrt. Mit Begleittext.

DM 19,80

114

Der Microcomputer im Kleinbetrieb, L. Oswald

Wie setzt man die modernen, preiswerten Microcomputer im Kleinbetrieb ein? Dieses Buch gibt Ihnen Aufschluß, wie Sie mit Microcomputersystemen in der Preisklasse zwischen DM 2.000,-- und DM 20.000,-- wichtige Aufgabe in Ihrem Klein- oder Mittelbetrieb bewältigen können. Inventuraufnahme, Inventurbewertung, Adresslisten, Rechnungen schreiben u. v. mehr. Erscheint Anfang 1981 **DM 39,80**

115

FOCAL Programmierhandbuch, L. Oswald

FOCAL ist eine Programmiersprache die von Digital Equipment entwickelt wurde. Sie ähnelt etwas der BASIC-Sprache, ist jedoch wesentlich leistungsfähiger. Hauptsächlich mathematische Probleme lassen sich mit ihr wesentlich einfacher lösen. Wie man diese Sprache auf einem 6502 Computersystem (KIM-1) mit Speichererweiterung implementiert, soll dieses Buch zeigen. Erscheint Mitte 1981 **DM 19,80**

116

Einführung 16-Bit Microcomputer, C. Lorenz

Eine kleine Einführung in die 16-Bit-Microcomputersysteme von Motorola, Zilog, Texas Instruments, Intel und Rockwell. Erscheint Mitte/Ende 1980 **DM 19,80**

117

FORTRAN für Heimcomputer, C. Lorenz

Einführung in die FORTRAN-Programmiersprache mit vielen Beispielen. Grundsätzliches über die verschiedenen Microcomputersysteme, die bereits mit FORTRAN-Compiler lieferbar sind. Allgemeine Übersicht, Tips und Hinweise. Mitte/Ende 1980 **DM 19,80**

118

Programmieren in Maschinensprache mit dem 6502, C. Lorenz und R. Lullus

Eine sehr ausführliche Beschreibung der 6502 Mnemonics mit kleinen Beispielen. Komplettes Assembler-Listing für CBM 8K alte ROMs und CBM 16, 32K neue ROMs. Text-Editor, Disassembler und Binder. BASIC-Listings mit genauer Beschreibung. **DM 98,--**

119

Programmieren in Maschinensprache (Z80), C. Lorenz

Eine sehr ausführliche Einführung in die Z80 Maschinensprache. Die Beispiele können mit Hilfe des TRS-80 Level II sowie dem T-BUG von TANDY und den T-BUG-Erweiterungen (IN LOCO, T-STEP, T-LEGS) ausgeführt werden. Ein unentbehrliches Buch für jeden, dem die BASIC-Programmiersprache von der Geschwindigkeit her zur Lösung seiner Aufgaben nicht mehr ausreicht. **DM 49,--**

120

Anwenderprogramme für TRS-80, M. Stübs

Ein Buch, voll mit interessanten Anwenderprogrammen für TRS-80 Level II 16K (teilweise Diskette und/oder Cassette). Hauptsächlich Programme für den Manager, Geschäftsmann, Klein- und Mittelbetrieb. Auch einige interessante Spiele sind enthalten. Terminkalender, Reservierungsprogramm für Omnibusunternehmen und Hotels usw.

DM 29,80

121

Microsoft BASIC-Handbuch, Microsoft

Die deutsche Übersetzung des erfolgreichen Microsoft BASIC-Handbooks. Leicht verständliche Einführung mit vielen interessanten Programmbeispielen.

DM 29,80

122

BASIC für Fortgeschrittene, C. Lorenz

Ein Buch für denjenigen, der tiefer in die BASIC-Programmiersprache einsteigen möchte. Sehr viele nützliche Unterroutinen, Programmiertricks und Hinweise.

DM 39,--

123

IEC Bus-Handbuch, von Max P. Gottlob

Ein Handbuch und Nachschlagwerk für alle Besitzer von Computern mit IEC (IEEE 488 Bus). Dazu gehört auch der PET sowie alle CBM-Computer. Grundlagen, das BUS-System, Meßdatenübertragung, Adressierung eines Instrumentes, kleines IEC-BUS-Lexikon u. v. a. ISBN 3 921682-73-8 **nur 19,80 DM**

124

Programmieren in Maschinesprache mit CBM

An Hand eines praktischen Beispiels (Sortierroutine) wird der Unterschied zwischen BASIC und Maschinenprogrammen gezeigt. Das Maschinenprogramm kann mit dem leistungsfähigen MONJANA/1 Monitor in ROM erstellt werden. Am Schluß finden Sie weitere wichtige Informationen wie Dez/Hex- Umrechnungstabelle, Befehlslisten, ASCII-Tabelle sowie eine ROM-Vergleichsliste zwischen 8k PET und den neuen CBM-Maschinen. 60 Seiten. ISBN 3-921682-70-3. **19,80 DM**

125

ELCOMP, Fachzeitschrift für Microcomputertechnik

ISSN-Nr. 0171-0958. Die kompetente Fachzeitschrift für das moderne Gebiet der Microcomputertechnik. Erscheint 10 x pro Jahr. Jahrespreis **DM 59,–** incl. MwSt. Porto und Verpackung. Wer die neuesten Informationen aus diesem Gebiet für sich nützen möchte, muß ELCOMP lesen. Technische Tips, Software, Bauanleitungen. Programmiertricks, Systembeschreibungen. Jeden Monat brandneu.

4,50 DM



ELCOMP Doppelheft

Von den zehn Heften pro Jahr erscheint im Juli/August und November/Dezember ein Doppelheft. **9,– DM**



Einführung in die Microcomputer Programmierung mit 6800

Eine sehr gute Einführung in die Microcomputertechnik mit Hilfe des Mikroprozessors 6800. Ausführliche Erklärungen mit vielen Beispielen und Anleitungen. Theoretische Grundlagen. CPU-Architektur, Befehlssatz, Systemaufbau. Hilfsmittel der Programmierung, Trainingsprogramme, Systemkomponenten, FIRMWARE. Ein komplettes Monitorprogramm (Betriebssystem) ist als Listing enthalten. Über 250 Seiten. ISBN-Nr. 3-921682-79-9 **49,– DM**



Programmieren mit dem CBM

Ein Hand- und Programmierbuch für alle CBM-Besitzer der 3000 und 8000er Serie. Viele Tricks und Programmierbeispiele, Anleitungen. Erscheint Ende 1980. **29,80 DM**



ELCOMP Leser Programmierhandbuch

Hier fassen wir die besten Programme unserer ELCOMP-Leser zusammen. Programme für PET, CBM, TRS-80, AIM, SUPERBOARD, C4P, EXIDY, SHARP, MZ80K, Apple II, NASCOM I u. II und TI 99/4 werden als Listing mit kurzer Beschreibung allen Lesern zugänglich gemacht. DIN A 4 ca. 300 Seiten. Erscheint Anfang 1981. **69,– DM**



Programmierbeispiele für CBM

Ein Buch mit vielen BASIC-Programmen für CBM und PET. Spiele, Geschäftsbereich, Erziehung und Wissenschaft, Utilities, trickreiche Programme, Hilfen für Maschinensprachenprogrammierung. Viele Programme für wenig Geld. Erscheint Ende 1980 **19,80 DM**



Cobol für Anfänger

Eine Einführung in die Cobol-Programmierung für den Microcomputer-Besitzer. Erscheint Anfang 1981. **19,80 DM**



CP/M-Handbuch

Grundlagen, Einführung, Hilfs- und Handbuch für jeden der mit dem "Software-Bus" arbeiten möchte. Ideal auch für Anfänger. Praktisches Handbuch für den Profi. Anfang 1980 **29,80 DM**



Welches Betriebssystem brauche ich?

Ein Leitfaden und Handbuch für den Anwender der gehobenen Microcomputerklasse. Anfang 1981. **19,80 DM**

Bücher in englischer Sprache aus dem Hofacker Verlag
Books in English, publushed by Ing. W. Hofacker GmbH

150

Care and Feeding of the Commodore Personal Electronic Transactor, edited by Silver Spur

Ein Buch für den PET-Besitzer und Hardware-Fan. Viele wichtige Kniffe und Tricks, die Sie benötigen, wenn Sie selbst ihren PET-Computer erweitern wollen. **DM 19,80**

151

8K Microsoft BASIC Reference Manual

Eine Einführung in die 8K Microsoft BASIC-Version mit vielen Beispielen. Dieses Buch braucht jeder, der einen Microcomputer mit Microsoft-BASIC besitzt. **DM 19,80**

152

Expansion Handbook for 6502 and 6800, S. Roberts

Viele interessante periphere Schaltungen für 6502 und 6800 Microcomputersysteme. Ein Buch das jeder benötigt, der sein System selbst erweitern möchte und noch mehr Vorteile daraus ziehen will. RAM-Platine, ROM-Platine, CPU-Platine, Ein-/Ausgabeplatine u. v. a. **DM 19,80**

153

Microcomputer Application Notes, Intel

Eine Zusammenfassung der interessantesten Applikationsberichte von Intel. Programmierbarer Interfacebaustein 8255, Anwendung des USART 8251, entwickeln mit statischen RAMs, Benutzung der seriellen Ein-/Ausgabeleitungen des 8085, Anwendung von 5V EPROMs, Anwendung des 2708 EPROMs u. v. **DM 29,80**

154

Complex Sound Generation using the SN76477

Dieses Buch zeigt Ihnen, wie man mit dem SN76477 die interessantesten und schönsten Töne erzeugen kann. Vogelgezwitscher, Dampfeisenbahn, Autorennen, Raketenabschuß, Granateneinschlag u. v. a. mehr. Viele Schaltungen mit genauen Beschreibungen. Kurzdaten. **DM 19,80**

155

The First Book of 80-US(TRS-80)

In diesem Buch haben wir noch einmal einige wichtige und interessante Programme sowie Artikel aus der amerikanischen Spezialfachzeitschrift für TRS-80 (80-US, The TRS-80 Users Journal) zusammengestellt und als Buch herausgegeben. Dieses Buch sollte nur kaufen, werdie Einzelhefte dieser Zeitschrift von Vol. I Nr. 1 bis Vol. II Nr. 5 noch nicht besitzt. Das Buch enthält sehr wertvolle, nützliche Informationen und phantastische Software: Kleinbuchstaben für TRS-80, Biorythmus für TRS-80, Bowlingspiel, Adressliste, Bildzeichnungsroutine, komplettes Geschäftsprogramm für Friseurläden, Telefon und Namensverzeichnis. Texteditor, Persönlichkeitsprofil u. v. a. **DM 19,80**

156

Small Business Programs, S. Roberts

Ein Buch für denjenigen, der die modernen Microcomputer (speziell TRS-80, APPLE, PET, North Star, Challenger) zur Rationalisierung in seinem Klein- oder Mittelbetrieb einsetzen möchte. Viele nützliche Tips, Hinweise und Programmierbeispiele. Dieses Buch sollte jeder Geschäftsmann und Microcomputerfreund besitzen. **DM 29,80**

157

The first Book of Ohio Scientific Vol. I

Das erste weltweit produzierte Buch für die erfolgreiche Ohio Scientific Challenger Computerserie. Grundlagen, viele Programmiertricks, Hardwaretips, Umbauanleitungen, Programmierbeispiele u. v. a. 186 Seiten. Glanzumschlag **19,80 DM**

158

The second Book of Ohio Scientific

Eingehende Beschreibungen über praktische und geschäftsorientierte Software. Speicher Test Programm, Tricks und Tips für Disketten-Anwender. Mini-Floppy-Expansion u. v. a. 159 Seiten. **19,80 DM**

159

The third Book of Ohio Scientific

Fortsetzung der oben beschriebenen Reihe. In diesem Buch findet der Hardware-Spezialist das was er schon länger gesucht hat. Wichtige Informationen für die Systemerweiterung, Schaltpläne u. v. a. 180 Seiten. Herbst 1980 **19,80 DM**

160

The fourth Book of Ohio Scientific

Ein Buch voll mit Programme für das Superboard, C4P, C4PMF und C28P. Die Softwarequelle für jeden Challenger Fan. Alle Programme sind getestet und auch auf Cassette verfügbar. 170 Seiten Listings u. Beschreibungen. Herbst 19 80.

Buch **29,80 DM**
Cassette (extra) **29,80 DM**

161

The fifth Book of Ohio Scientific

Frühjahr 1981. **19,80 DM**

162

ATARI GAMES IN BASIC

Ein Büchlein mit vielen Programmen für den ATARI-Computer (400 + 800). 64 Seiten. Ende 1980 **19,80 DM**

163

The peripheral Handbook

Ein Handbuch über Drucker, Floppys, Hard-Disks, und alles was Sie um den Micro-Computer herum benötigen. Tricks und Kniffe. Technische Beschreibung. Anfang 1981. **29,80 DM**


–INHALTSVERZEICHNISSE

Inhaltsverzeichnis der zurückliegenden ELCOMP - Hefte

Ein einfaches PROM-Programmiergerät
Ein-/Ausgabeprogrammierung mit PET
Erweiterung für NASCOM-1
Lernen für die Anwendung von Microprocessoren

Nr. 9 September 4,50 DM
Tips für den BASIC-Programmierer
Wie lernt man BASIC, Teil X
EUROCOM-1
Michrochatter
Die neuen ROMs im CBM
Neue Publikationen
Speichererweiterung für den PET
Musikprogramm für den NASCOM-1
Computersystem 79
Umschalten des Bildschirms auf Breitschrift beim TRS-80 Level I
Vergleich dreier Zahlen
ELCOM-Leserbriefe
Diskettenpower APPLE II
Neuer Einplatinencomputer von Motorola
Einstellung der Tonspur beim PET

Nr. 10 Oktober 1979 4,50 DM
Ein-/Ausgabeprogrammierung mit dem TRS-80
Texas Instruments stellt seinen Heimcomputer vor
Microchatter
Wie lernt man BASIC, Teil XI
Der MZ-80 von Sharp
Microcomputer als Hobby
Ein komplettes Textverarbeitungssystem mit CBM-Computer u. Heathkit-Drucker
Wer ist OHIO SCIENTIFIC ?
KURDIS - Ein Programm zur Kurvendiskussion
BASIC im Unterricht
Monolithischer 8 Bit Analog/Digital-Wandler
Band-Test

Nr. 11/12 Nov./Dez. 1979 9,00 DM
Jetzt kommt PASCAL
Aus der PET-Modulkiste
HISTORGRAMM
32k RAM Zusatzspeicher f. PET 2001
Exidy Sorcerer
Kleinschrift und Verbesserung des Video-RAMs beim TRS-80
Filmkamerasteuerung mit PET
Anschluß eines Floppy-Disk-Controllers an 6500 Microprozessoren
Anschluß eines Baudot-Fernschreibers an einen Microcomputer
Ein Besuch bei ATARI
Kalender
Ein wenig Trigonometrie
Berechnen der Kreiszahl
Der NASCOM-2 ist da
PET als IC-Tester
Das NIM-Spiel und was es damit auf sich hat
Linking Loader für PET 2001 (8k)
MEMDMP-Hex Dump- Programm für 6502 Systeme
KIM-1 schreibt auf Skop
IRRGARTEN
KIM-Orgel
Lichtgriffel
Meldeton für PET
Preiswerter Drucker f. NASCOM-1
Wie lernt man BASIC, Teil XII

Bereits vergriffen !

Nr. 1 Januar 1980 4,50 DM
Microcomputer-Software und Vernunft
TRS-80 als "elektronisches Kurvenlineal"
Schrottknüppel
Programmbeschr.: FLAG OPERATION
Aus der PET-Modul-Kiste
Alternatives TINY-BASIC für den TRS-80 unter Level 1
Programmtricks APPLE II
NEU!!! Einführungskurs: Programmieren in Maschinensprache auf Z-80
Programmiertrick für TRS-80
Eine professionelle Analogschnittstelle GRAPHSYS
Wie lernt man BASIC, Teil XIII
Programmierhilfen f. NASCOM-1
SUMA 85 (Selbstbausystem)
Automatische Zeilennummerierung für PET
ULTRA-MON-CBM Bedienungsanleitung
Entwicklungsboard f. AmZ8000
ELTRO-HOBBY 79, der große Renner

Nr. 2 Februar 1980 4,50 DM
"Echte" Großschrift beim TRS-80
BASIC-Programme - kürzer und schneller
Datenlogger BASIC - Programmierbar
EPROM-Programmiergerät für PET
HIGH-Resolution-Graphic auf dem APPLE II
Wie lernt man BASIC, Teil XIV
Der TI-Programmer
SUMA 85
Programmieren in Maschinsprache mit Z-80
PET-Petits

Nr. 3 März 1980 4,50 DM
Der Personalcomputer von HP ist da
Lernen mit dem AIM, Teil 1
Die neuen CBM-Modelle
SUMA 85, Teil 3
Programmieren in Maschinensprache mit Z-80, Teil 3
Maschinencodemonitor in TINY-BASIC
PET-Petits
Was ist Datenschutz?
Monitorprogramm für SC/MP
Wie lernt man BASIC, Teil XV
Quadratische Gleichungen
Wer ist Jim Butterfield?
Messebericht: Comsumer Electronics Show
Der PC-100

Nr. 4 April 1980 4,50 DM
Grundlagen der Finanzbuchhaltung für den Microcomputeranwender
SUMA 85, Teil 4
Messebericht: Hobbytronic Dortmund
Berechnung ganzrationaler Funktionen
Aus der PET-Modulkiste
SDS-100 MicroMini
Erstes Home-Computer-System von TI
Programmieren in Maschinensprache mit Z-80, Teil 4
PRINT USING für X80 und PET
Microchatter
Wie lernt man BASIC, Teil XVI
PET Petits
Grafik mit dem Superboard
Untersuchen von Texten auf die Häufigkeit des Auftretens einzelner Buchstaben
Zum lernen ideal. . . .
Hochinteressante Interviews

Nr. 5 Mai 1980 4,50 DM
Aus der PET-Modul-Kiste
Z80-Disassembler für den NASCOM
Implementierung einer REAL-TIME-CLOCK
SUMA 85, Teil 5
Programmieren in Maschinensprache mit Z-80, Teil 5
Lernen mit AIM, Teil II
AIM 65 hilft im Büro
Protokollieren von TRS-80 Bildschirmausgaben
Wie lernt man BASIC, Teil XVII
West Coast Computer Faire
Der HY Q-1000
Der $BASE_2$ Drucker

Nr. 6 Juni 1980 4,50 DM
Die einfache Verkettungstechnik
Programmieren in Maschinensprache mit Z-80, Teil 6
Lernen mit dem AIM, Teil 3
Microcomputer Fools Corner
Challenger C4PMF Microcomputer
SUMA 85, Teil 6
Wie lernt man BASIC, Teil XVIII
TINY-FORTH für TRS-80 Level II, 16k
CBM-Disk BASIC Version 4.0
Heathkit - Matrixdrucker / Erfahrungsbericht
Monostabile Kippstufen
5. West Coast Computer Faire
Hex-DUMP in BASIC für Superboard
Michrochatter

**ELCOMP ist in der Microcomputer-Berichterstattung immer vorne.
Keiner, der mitreden will, kann es sich noch leisten, ELCOMP nicht zu lesen! Bestellen Sie deshalb heute alle noch verfügbaren Hefte und sichern Sie sich ELCOMP im Abonnement für die Zukunft.**

Hiermit gebe ich folgende Bestellung unter Anerkennung Ihrer Lieferungs- und Zahlungsbedingungen auf.



Menge	Artikelbezeichnung	Einzelpreis

Menge	Artikelbezeichnung	Einzelpreis

Unterschrift (für Jugendliche unter 18 J. der Erziehungsberechtigte) Datum

BESTELLKARTE für ein ELCOMP-Abonnement

Auslands-Abonnement (Europa) + DM 10.-- Versandkosten

- O Bitte senden Sie mir ab sofort alle künftigen Ausgaben der ELCOMP. Preis DM 59,-- pro Jahr incl. Versand und Verpackung. Lieferung ab Ausgabe..............................
- O Bitte senden Sie mir alle bereits erschienen Ausgaben zum Sonderpreis DM 37,-- (Sept. 78 – Okt. 79, außer Nr.3/4 1978, Nr. 2, 4,11/12 1979)
- O Jahrgang 1980 DM 49,–
- O Ich bezahle im voraus für..........Jahr(e) DM.................., Pschk. München 15994807, Dresdner Bank München Kto. 5390 370, BLZ 700 800 00.
- O Ich bitte um Lieferung auf Rechnung.
- O Ich bitte um Abbuchung von meinem Konto (Giro od. Postscheckkonto)

Kto.Nr.. BLZ...

Geldinstitut:.................................... Ort:...

Name/Vorname

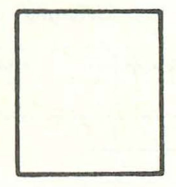
Straße / Hs-Nr.

PLZ Ort

Datum.. Unterschrift..

ELCOMP

POSTKARTE



Absender
Bitte deutlich ausfüllen

____
Vorname/Name

____
Beruf

____
Straße/Nr.

____
Plz Ort

ELCOMP

MIKROCOMPUTER BOOK STORE

Tegernseerstr. 18

D-8150 Holzkirchen /Obb.

ABSENDER:

....
Name, Vorname

....
Straße

( ) ....
PLZ Ort

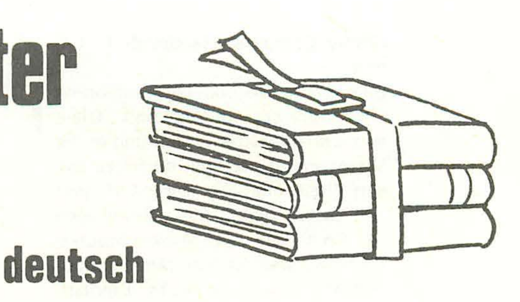
....
Telefon

Das Abonnement wird automatisch verlängert
(Kündigung 8 Wochen z. Abonnement-Ablauf)



Ing. W. Hofacker GmbH
Tegernseer Straße 18
D-8150 Holzkirchen

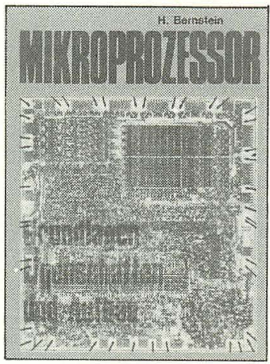
Mikrocomputer Fachbücher



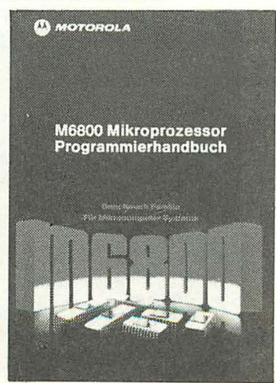
deutsch



Aktiv Training Microcomputer 8080 8085
Dieses Werk mit über 360 Seiten beschäftigt sich ausschließlich mit den Microcomputerbausteinen und Peripherieelementen der 8080A und 8085 Microprozessoren. Grundlagen, Einführung 8080 Microcomputer, Programmieren in Assembler, PL/M Compiler, Speicherbausteine u. v. a. mehr. Ideal für jeden, der ein System mit 8080, 8085 CPU besitzt. Auch der Z-80 Systembesitzer kann von diesem Buch viel profitieren. Am Schluß des Buches finden Sie noch ca. 150 Programmlistings in Maschinensprache (Nützliche Utilities und Spielprogramme).
Best.-Nr. 30 DM 49,80



Mikroprozessoren, Eigenschaften und Aufbau, Teil 1, H. Bernstein
Grundlagen, Eigenschaften und Aufbau von Mikroprozessoren, Organisation von Recheneinheiten und Mikroprogrammen. Programmierung und Klassifizierung von Mikroprozessoren. Ablaufdiagramm, Flußdiagramm. Ein-Chip-Technik und Multi-Chip-Technik, Transfer- und Sprungfunktionen. Speichertechnik: RAMs, ROMs, FIFO, FILO. Programmierbare logische Arrays (PLA). Anwendungsbeispiele und Anwendungsbereiche. Über 120 Seiten.
Best.-Nr. 22 DM 19,80



M6800 Mikroprozessor Programmierhandbuch
Ein deutschsprachiges Handbuch für den 6800 Microcomputer. Grundlagen, Einführung und genaue Befehlserklärungen.
Best.-Nr. 8063 DM 19,80

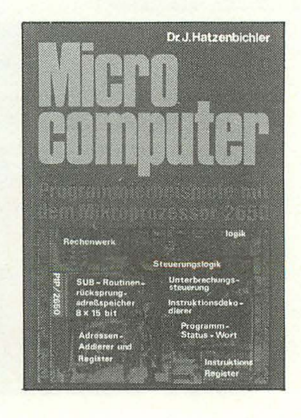
Hobby Computer Handbuch, C. Lorenz
Eine leicht verständliche Einführung in die Mikrocomputertechnik. Diese sehr umfangreiche Einführung in die Microcomputertechnik dürfte zu diesem Preis einmalig sein. Auf über 450 Seiten finden Sie– Grundlagen der Computer- und Microcomputer-Technik, was ist ein Microcomputer? Microcomputer KITs, Einplatinencomputer, CAT, OSI, POLY 88 u. v. a. Das ideale allumfassende Buch für den Microcomputertechniker. Für Industrieanwendung ebenso geeignet wie für den Hobby-Computer-Fan. Über 450 Seiten.
Best.-Nr. 25 **DM 29,80**





Mikrocomputer Programmierbeispiele für 2650, Dr. J. Hatzenbichler
Eine Einführung in die Programmierung von Mikrocomputern anhand des Prozessors 2650 von Signetics. Viele Programmierbeispiele in Maschinensprache, die Sie auf einem preiswerten Mikroprozessorsystem MIKIT 2650-P2 ausführen können. Zeitschleifenprogramme, Blinkschaltung, Lauflicht, Stufenzähler, Elektronischer Würfel, Stopuhr, Reaktionszeittester, Computer Musik Programm u. a. Zu diesem Buch ist auch ein komplett aufgebautes und getestetes Mikrocomputersystem erhältlich, auf dem Sie alle beschriebenen Programme selbst ausführen können. Über 120 Seiten.
Best.-Nr. 33 **DM 19,80**

Microcomputertechnik
Z-80, Z8, Z8000 v. H. P. Blomeyer-Bartenstein
völlig neu überarbeitete Auflage. In diesem Buch finden Sie eine umfassende, einführende und weiterführende Hilfe zum Einstieg in die Microcomputertechnik mit vielen Schalt- und Programmierbeispielen. Als praktische Betrachtungsgrundlage dient das supermoderne Microcomputerkonzept Z80A von Zilog. Das Buch geht auf alle wichtigen Zusammenhänge ein und erklärt diese dem Leser so ausführlich, daß kaum noch Fragen offen bleiben. Über 240 Seiten. **Best.-Nr. 24** **DM 29,80**





Programmieren in Maschinensprache mit 6502

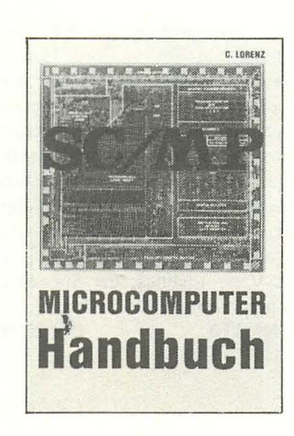
Ein deutsches Buch für jeden, der mit dem 6502 Mikroprozessor arbeitet. Grundlagen Maschinensprache, Beschreibung des Extended Monitors für Superboard von Challenger. Ausführliche Beschreibung jedes einzelnen Maschinenbefehls mit Beispiel. Übersichtliche Beschriftung. Sie finden jeden Befehl schnellstens und können nachlesen, was im Computer selbst geschieht. Am Schluß finden Sie Listings und Beschreibung für einen kompletten 2-Pass-Assembler, Editor, Binder und Disassembler. Für PET 8K, alte ROMS und für die neue CBM-Serie.

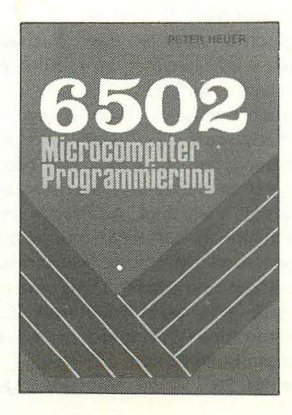
Best.-Nr. 118 **DM 98,--**

Handbuch für SC/MP

Ein echtes Handbuch für SC/MP (INS8060) Microcomputerbesitzer und solche, die es werden sollen. Komplette Einführung mit vielen Schaltbeispielen, Schaltbildern und Programmlistings in Maschinensprache und TINY BASIC. Komplettes ELBUG-Listing, CPU-Karte, RAM-Karte, ROM-Karte, ROM-Programmierer, Cassetten-Interface, Fernsehinterface, Netzteil, Hex Ein-/Ausgabe, SC/MP Einkartenmicrocomputer u. v. a. mehr.

Best.-Nr. 108 **DM 29,80**

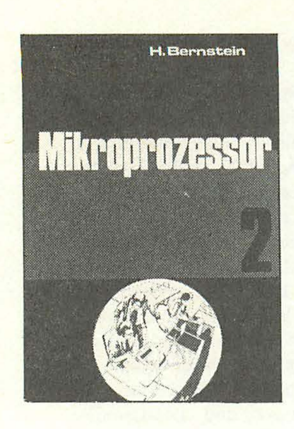




6502 Microcomputer-Programmierung, Peter Heuer

Eine deutschsprachige Einführung in die Maschinensprachenprogrammierung anhand des 6502 Microcomputers. Ein echtes Anleitungsbuch zum Einstieg in die Microcomputertechnik mit Hilfe des KIM-1. Viele Programmierbeispiele, die von einem Pädagogen speziell für Anfänger entwickelt wurden. Auch PET, AIM, SYM und ATARI-Besitzer brauchen dieses Buch.

Best.-Nr. 109 **DM 29,80**



Mikroprozessor, Teil 2
von H. Bernstein. Die Fortsetzung unseres ersten, so erfolgreichen Buches über Mikroprozessoren. Technologie von Mikroprozessor- und Speicherbausteinen. Festwertspeicher, PROM, REPROM, FIFO, Schieberegister, MPR-Register, ARL-Register, SAR-Register. Aufbau eines Mikroprozessorsystems mit 8080, RAM- und ROM-Schnittstellen. Befehlssatz 8080. Über 120 Seiten.
Best.-Nr. 26 DM 19,80

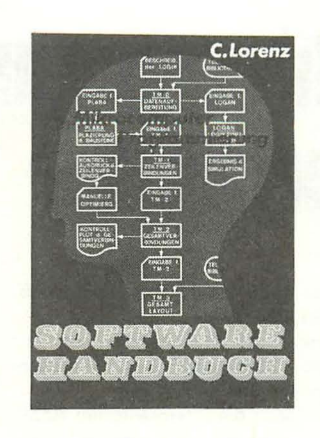
Programmierhandbuch für PET
C. Lorenz
Ein vom Hofacker-Verlag für den PET produziertes Buch. Es beginnt da, wo Ihr mitgeliefertes Handbuch aufhört. Einführung Maschinenprogrammierung, Assembler, Ein-/Ausgabeprogrammierung, Programmiertricks, Analog/Digital-Wandler, Graphik, Spracherkennung u. v. a. mehr. Viele Listings, die Sie selbst eintippen können. 324 Seiten.
Best.-Nr. 110 DM 29,80





Microcomputer Lexikon und Wörterbuch von A–Z, C. Lorenz
Englisch/Deutsch – Der Fachausdruck wird übersetzt, ausführlich erklärt und erläutert.
Deutsch/Englisch – Übersetzung des Fachausdrucks. Ein Hilfs- und Arbeitsbuch für jeden, der sich heute mit der modernsten Elektronik beschäftigt. Viele engl. Ausdrücke werden heute in der Elektronik, Computer- und Mikroprozessortechnik verwendet und oft fehlt uns eine genaueste und präzise Erläuterung. Ein Lexikon und Wörterbuch in einem einzigen Buch vereint.
Best.-Nr. 28 DM 29,80

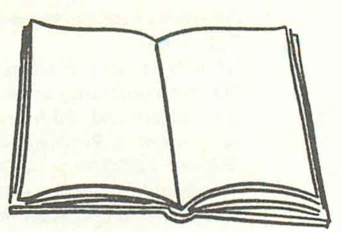
Mikroprozessor Software Handbuch
C. Lorenz
Grundlagen und Einführung in die Mikrocomputerprogrammierung, Grundlagen und Einführung in die wichtigsten Programmiersprachen (BASIC, FORTRAN, ASSEMBLER-Sprachen). Zusammenstellung der wichtigsten Befehlslisten: 8080, Z80, M6800, National, Fairchild etc. Ein Software Handbuch für jeden, der mit Mikroprozessoren oder Mikrocomputern zu tun hat. Über 200 S.
Best.-Nr. 27 DM 29,80



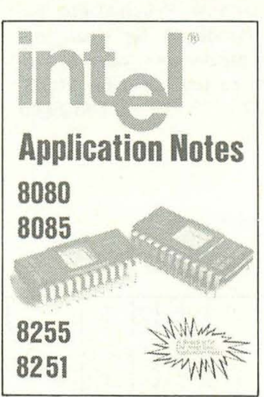
Best. Nr.	Stck.	BUCHTITEL BEZEICHNUNG	a DM	ges. DM
1		TBB – Handbuch Band 1	19.80	
2		TBB – Handbuch Band 2	19.80	
3		Elektronik im Auto	9.80	
4		IC – Handbuch, TTL, CMOS, Lin.	19.80	
5		IC – Datenbuch, TTL, CMOS, Lin.	9.80	
6		IC – Schaltungen, TTL, CMOS	9.80	
7		Elektronik Schaltungen	5.–	
8		IC – Bauanleitungshandbuch	19.80	
9		Feldeffekttransistoren	5.–	
10		Elektronik und Radio 4. Auflage	19.80	
11		IC – NF – Verstärker	9.80	
12		Beispiele Integrierter Schaltungen	19.80	
13		HEH, Hobby Elektronik Handbuch	9.80	
14		IC – Vergleichsliste	9.80	
15		Optoelektronik Handbuch	19.80	
16		CMOS Teil 1, Einführung	19.80	
17		CMOS Teil 2, Schaltbeispiele	19.80	
18		CMOS Teil 3, Schaltbeispiele	19.80	
19		IC – Experimentier Handbuch	19.80	
20		Operationsverstärker	19.80	
21		Digitaltechnik Grundkurs	19.80	
22		Mikroprozessoren 2. Auflage	19.80	
23		Elektronik Grundkurs	9.80	
24		Mikrocomputer Technik	29.80	
25		Hobby Computer Handbuch	29.80	
26		Mikroprozessor Teil 2	19.80	
27		Mikroprozessor Software Handbuch	29.80	
28		Lexikon + Wörterbuch	29.80	
29		Microcomputer Datenbuch	49.80	
30		Aktivtraining Microcomputer	49.80	
31		57 Programme in BASIC	39.–	
32		Circuits Digital Et Pratique	19.80	
33		Microcomputer Programmierbeispiele	19.80	
41		IC-KIT Experimentierplatine WH-1 g	79.–	
105/1		TTL-Experimentierbuch	5.–	
106/1		CMOS-Experimentierbuch	5.–	

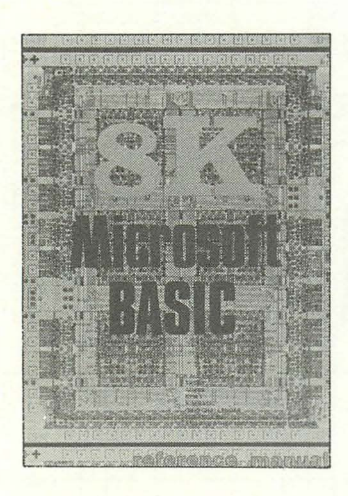
Mikrocomputer Fachbücher

englisch



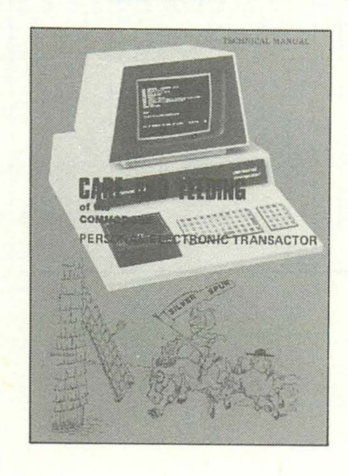
Intel Application Notes (8080,8085, 8255, 8251)
Dieses Buch braucht jeder, der mit 8080, 8085 oder Z-80 Mikroprozessoren arbeitet.
Wir haben die interesantesten Applikationsberichte in diesem Buch zusammengefaßt.
Aus dem Inhalt: Designing with Intel's Static RAMs 2102, Memory Design with the Intel 2107B.
8255 Programmable Peripheral Interface Applications, Using the 8202 Dynamic RAM Controller u. v. a.
Best.-Nr. 153 DM 29,80



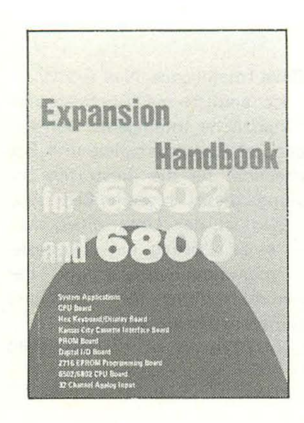


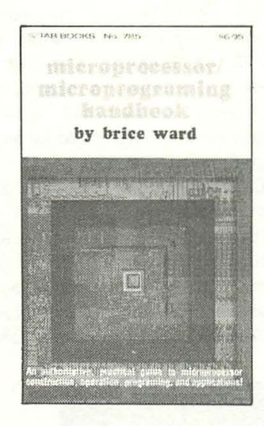
Microsoft 8K BASIC Reference Manual
Eine sehr gute BASIC-Einführung. Auch als Handbuch zum Nachschlagen bestens geeignet. Ideal für jeden PET, CBM, TRS-80, KIM-BASIC, SYM-BASIC, AIM- und APPLE-Besitzer. 73 Seiten DIN A4 mit vielen Beispielen. Eine Produktion des Hofacker-Verlages in englischer Sprache.
Best.-Nr. 151 DM 19,80

Care and Feeding of the Commodore PET
Das ideale Buch für den Hardware-Bastler. Viele Tricks, Schaltbilder, Hinweise und Erläuterungen für den, der gerne selbst Erweiterungen bauen möchte. Memory Map für 8K PET und CBM, Bauanleitung für eine serielle Schnittstelle u. v. a. mehr.
Best.-Nr. 150 DM 19,80



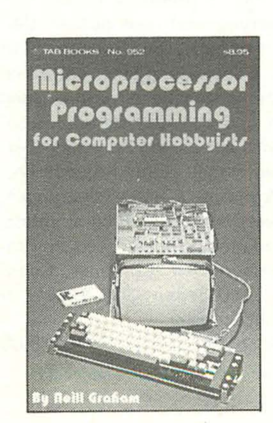
Expansion Handbook for 6502 and 6800
Das ideale Handbuch für alle KIM, SYM, AIM, PET und Challenger Computer-Freunde. Das Buch beschäftigt sich ausschließlich mit dem S-44-Bus. Dies ist exakt der Bus von SYM, AIM und KIM. Sehr viele Schaltbilder: CPU-Platine, Hex-Tastatur Eingabe, Kansas City Interface, RAM u. ROM-Karte, Analog-Eingabe Board u. v. a. Das Buch ist für jeden 6502 Systembesitzer unentbehrlich. Ca. 150 Seiten
Best.-Nr. 152 **DM 19,80**



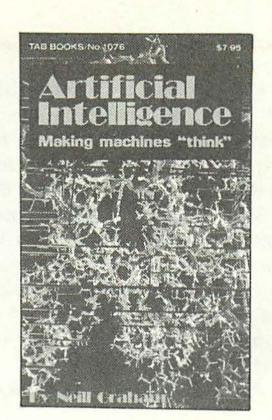


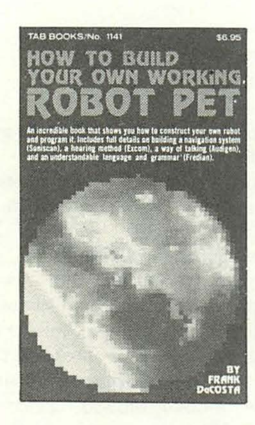
Microprocessor/Microprogramming Handbook, Brice Ward
Ein praktisches Handbuch für jeden, der sich mit Mikroprozessoren beschäftigen möchte. Auf über 290 Seiten finden Sie Grundlagen, alles über Programmierung und Anwendungsbeispiele der interessantesten integrierten Schaltungen. Inhalt: Einführung in die Mikroprozessortechnik. Der Mikroprozessor von innen. MCS4, MCS40, MCS80, Speichersysteme, Mikroprogrammierung in Maschinen- und Assemblersprache.
Best.-Nr. 785 **DM 35.--**

Microprocessor **Programming for the Computer Hobbyist**
Dieses buch ist speziell für den Computer-Hobbyisten geschrieben, der sich bereits mit weiterführenden Programmiertechniken und Datenstrukturen beschäftigen möchte. Inhalt: Höhere Programmiersprachen (PL/M,PL/1), arithmetische Funktionen, Suchprogramme werden im Zusammenhang mit dem Schachspielproblem behandelt. Über 380 Seiten in englischer Sprache
Best.-Nr. 952 **DM 39,--**



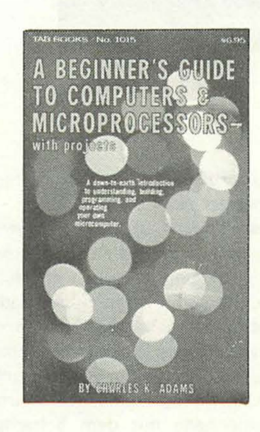
Artificial Intelligence, Neil Graham
Hier ist endlich ein Buch speziell über künstliche Intelligenz. Der Autor benutzt Computerspiele und Robotertechnik für die illustrative Behandlung dieses brandheißen Themas. Was ist der Unterschied zwischen einer Lösung einer komplexen Programmieraufgabe und der Programmerstellung für intelligente Computerentscheidungen.
Best.-Nr. 1076 **DM 29,80**



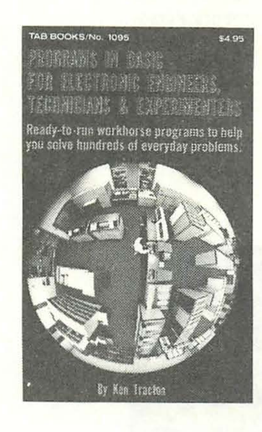


How to build your own working ROBOT PET
An incredible book that shows you how to construct your own robot and program it. Includes full details on building a navigation system (Soniscan), a hearing method (Excom), a way of talking (Audigen) and an understandable language and grammer (Fredian).
Eine wertvolle Hilfe für alle, die sich einen Roboter bauen wollen. (8085) CPU). Viele wertvolle Schaltungen, die Sie im Zusammenhang mit TRS-80 oder 8085-Computern verwenden können. 238 Seiten.
Best.-Nr. 1141 **DM 29,80**

Beginner's Guide to Computer Programming, Brice Ward
Eine ideale Einführung in die Programmierung von Computern (Microprozessoren). Das Buch beginnt mit der Entwicklung einer einfachen Programmiersprache für den eigenen Bereich und zum Selbststudium, sodann wird auf andere Sprachen übergegangen. Inhalt: Grundlegende Programmkonzepte, I/O-Schaltungen, Flußdiagramme, Programmtest, Schleifen, Indexregister, versch. Programmiersprachen, Compiler, Cobol u. v. a., 480 Seiten, 364 Bilder , in englischer Sprache.
Best.-Nr. 1015 **DM 29,80**



PROGRAMMING MICROPROCESSORS, M.W.McMurran
Dieses Buch reicht von der grundlegenden Microprozessororganisation über Zahlensystem, Flußdiagramme, Adressierung, Assemblierung über Subroutinenbeschreibungen, Programmierhilfen, Datenaustausch, Compilern bis hin zu speziellen Programmiertechniken. Schwergewicht Software. 270 Seiten, in englischer Sprache
Best.-Nr. 985 **DM 35.--**



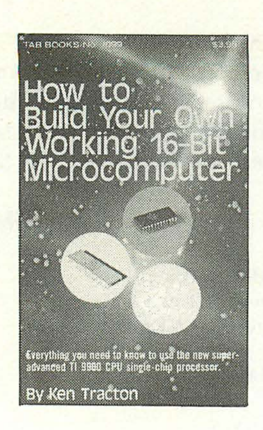
25 Games in BASIC (Listings). Die Programme laufen mit kleinen Änderungen auf den meisten Microcomputern. Inhalt: Number, Letter, Stars, Trap, Bagels, Mugwump, Hurkle, Snark, Reverse, Button Chomp, Taxman u. v. a.
Best.-Nr. 8057 **DM 9,80**

Digital Interfacing with an Analog World, Joseph J. Carr
Wie entwirft und baut man Interfaceschaltungen für Microcomputer? Der Schwerpunkt liegt bei der Verbindung mit der analogen Welt. Viele Tabellen, Schaltungen, Berechnungen und Hinweise. Über 400 Seiten.
Best.-Nr. 1070 **DM 39,--**



Complex Sound Generation with SN 76477
Ein Applikationsheft für einen der interessantesten integrierten Bausteine unserer Zeit. Ein LSI-Baustein zur Tonerzeugung. Je nach äußerer Beschaltung können Sie mit diesem Baustein die verrücktesten Töne erzeugen.
Dampfeisenbahngeräusch mit Dampfpfeife, Vogelgezwitscher, Hundegebell, elektronische Orgel, Schuß mit Explosion u. v. a. mehr.
Best.-Nr. 154 **DM 19,80**

DR.DOBB'S JOURNAL OF COMPUTER CALISTHENICS AND ORTHODONTIA, VOLUME ONE
Die komplett gebundene Sammlung der ersten sehr interessanten Ausgaben von Dr.Dobb's Journal. Inhalt: Entwickeln Sie Ihre eigene BASIC-Sprache, Entwicklungshilfen für Tiny Basic, Wie programmiere ich den ALTAIR 8800 für Computer-Musikstücke, kompl.Listing des Denver Tiny-BASIC ' Computer für Musik und Komposition, Hardware und Software für Sprachsynthese, MINOL - ein Tiny Basic mit String, in 1,75 k Byte, System Monitor für 8080, Palo Alto Tiny Basic, kompl.Listing, 8080 Text Editor, Tiny Trek-Game, Fließkommaroutinen für 6502, Comp.Music, Z 80 KIT, 6800 Assembler, u.v.a., ca. 260 Seiten A 4
Best.-Nr. 80/20 **DM 59.--**



How to Build your Own Working 16-Bit-Microcomputer von Ken Tracton

Alles, was Sie über den neuen Supermicroprozessor TI9900 wissen müssen. Beschreibung des Schaltkreises, Peripherie-Bausteine für TMS9900. Tricks, Systemaufbau etc. Sehr interessant, da der TMS9900 das Herz des neuen Texas Instruments Personal Computers ist.

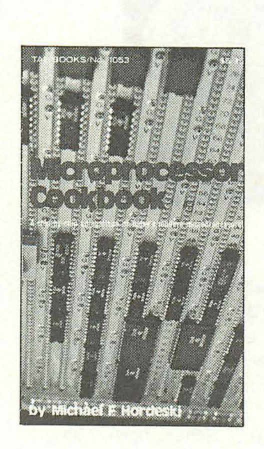
Best.-Nr. 1099 **DM 14,80**

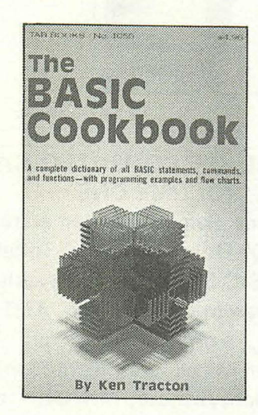
Microprocessor Cookbook, von Michael Hordeski

A chip-by-chip-comparison of todays modern microprocessors.

Die wichtigsten Prozessortypen werden genau beschrieben (Schaltung, Blockdiagramm, Befehlslisten, Programmbeispiele) und verglichen. 8080, 6800, F8, Z80, TMS 9900, SC/MP, Bit Slices, R6500, 264 S.

Best.-Nr. 1053 **DM 24,80**



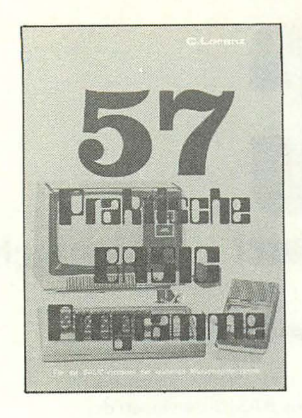


The BASIC Cookbook, von Ken Tracton

Ein komplettes Dictionary mit BASIC-Befehlen. Alle Befehle sind in alphabetischer Reihenfolge geordnet und jeder Befehl genau beschrieben. Zu jedem Befehl ist ein Demonstrationsbeispiel beigefügt.

Dieses Buch braucht jeder BASIC-Programmierer!

Best.-Nr. 1055 **DM 24,80**



57 praktische BASIC-Programme
C. Lorenz, Ken Tracton
Ein Buch mit technisch-wissenschaftlichen Programmen und einer großen Anzahl von Spielprogrammen in BASIC. (Games). Ein Buch für jeden, der sich mit dem faszinierenden Hobby der Microcomputertechnik befassen will. Alle Listings sind in BASIC und können auf den meisten Personal Computer Systemen gefahren werden. Alle Programme wurden sorgfältig getestet. Zum Beweis ist für jedes Programm ein Protokoll des Probelaufs abgedruckt.
Best.--Nr. 31 **DM 39,--**

The A to Z Book of Computer Games
26 aufregende und instruktive Spielprogramme fix und fertig zum Eingeben in jeden BASIC-Computer. Nach einer kleinen Einführung in die Programmierung folgen interessante Spielprogramme, die jeder Microcomputerbesitzer gesehen haben muß.
(Bandit, Cokes, Dice, Elevate, Firecard, Gunners, Hobbit, Invert, Justluck, Knights, Lapides, Match, Naughts + Crosses, Otello, Par 2, Quantal, Roulette, States Twenty 1, Ultranim, Verboten, Wumpus, X-Change, YATC, END.
Am Schluß werden noch einige Tips zur Umwandlung in andere BASIC-Versionen gegeben.
Über 300 Seiten.
Best.-Nr. 1062 **DM 29,80**

englisch

24 Tested Ready to RUN Game Programs in BASIC, Ken Tracton
Spaß- und Spielprogramme in BASIC. Viele Programme enthalten spezielle Anpassungshinweise an die Homecomputertypen TRS-80 und PET. Viele Graphik- und Zeichenprogramme. Ideal für jeden BASIC-Computer-Besitzer.
Best.-Nr. 1085 **DM 24,80**







25 Games in BASIC (Listing) Die Programme laufen auf dem PET. Inhalt: Number, Letter, Stars, Trap, Bagels, Mugwump, Hurkle, Snark, Reverse, Button, Chomp, Taxman u. v. a.
Best.-Nr. 80/57 **9,80**

57 Practical Programs and Games in BASIC, ca. 200 Seiten Listings.
Best.-Nr. 1000 **DM 35,--**

BASIC Software Library Vol. I, Personal Bookkeeping, Games and Pictures, 49 BASIC-Programme.
Best.-Nr. 80/50 **DM 99,--**

BASIC Software Library Vol. IV, General Purpose und viele Spiele, 21 BASIC Programme.
Best.-Nr. 80/53 **DM 39 --**

Dr. DOBBs COMPUTERZEITSCHRIFT. Die komplett gebundene Sammlung der ersten, sehr interessanten Ausgaben. Sehr viele Programmlistings für den Hobbycomputerfan.
Best.-Nr. 80/20, ca. 260 Seiten DIN A 4 **DM 59,--**

Microprocessor Software Handbuch, C. Lorenz
Microcomputerprogrammierung für Anfänger und Fortgeschrittene. BASIC-Einführung in Deutsch.
Best.-Nr. 27, über 280 Seiten **DM 29,80**

PET 6502 Programmierhandbuch, das Handbuch für jeden PET-Besitzer und solche, die es werden wollen. Einführung, Programmbeispiele, PET-Erweiterungsmöglichkeiten.
Best.-Nr. 110 **DM 29,80**

The BASIC-Cookbook, Ken Tracton
Über 138 Seiten BASIC-Einführung mit vielen praktischen Programmierbeispielen. Alphabetisch geordnet, ideal auch als Lexikon.(Diagnose, Erhöhung der Rechengeschwindigkeit u. v. a.)
Best.-Nr. 1055 **DM 24,80**

A Beginner's Guide to Computers & Microprocessors – with projects
Eine Einführung von Anfang an in die Mikroprozessortechnik. Bauen Sie Ihren eigenen Computer, programmieren Sie ihn und bedienen Sie ihn. Viele praktische Schaltungen. Über 300 Seiten.
Best.-Nr. 1015 **DM 29,80**

650X Software Manual (Programming Manual), sämtliche wichtige Daten und Programmierhinweise für die 6502 Programmierung. Ca. 200 Seiten.
Best.-Nr. 80/42 **DM 19,80**

650X Hardware Manual, 6502 Systemorganisation, Grundlagen, Applikationshilfen u. v. a. mehr.
Best.-Nr. 80/43 **DM 19,80**

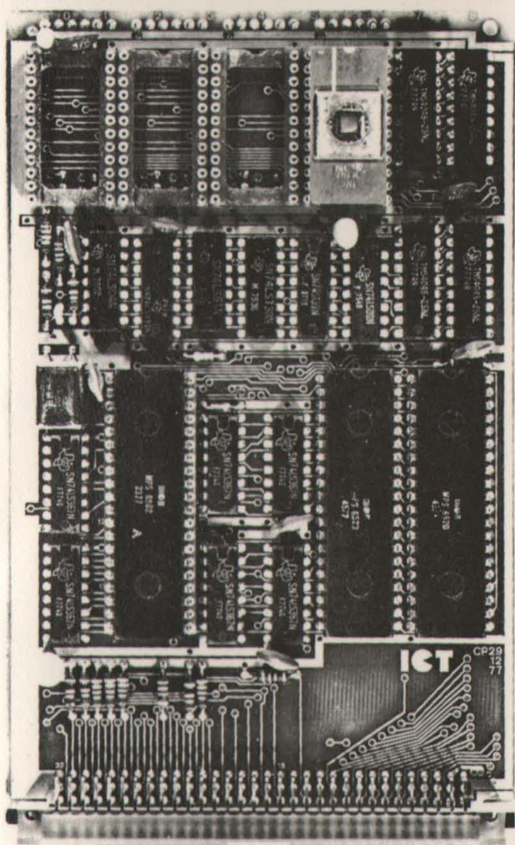
6500 Seminarheft (Briefing Notebook), eine kleine Einführung in die 6500 Systemfamilie.
Best.-Nr. 80/45 **DM9,80**

6500 Datenblätter, eine Sammlung aller wichtigen Datenblätter aus der 6500-Familie.
Best.-Nr. 80/46 **DM 9,80**

6502 Microcomputer, Aufbau und Programmierung. Ein deutsches Anleitungsbuch zum Einstieg in die Microcomputertechnik mit Hilfe des KIM-1. Viele Programmierbeispiele von einem Pädagogen speziell für den Anfänger entwickelt. (PET, VIM und AIM)
Best.-Nr. 109 **DM 29,80**



Mikrocomputer-Zentraleinheit ESCO-1



Zentraleinheit ESCO-1 mit Mikroprozessor 6502A, zwei PIAs 6520, 2K Bytes RAM und EPROM 2716 sowie drei IC-Sockeln für ROMs bzw. EPROMs.

Technische Daten und Abmessungen

Abmessungen	Europa-Format 100 x 160 mm
Bauhöhe	15 mm max. ≙ 3 TE
Steckverbindung	indirekt, 96-polig DIN 41 612
Taktfrequenz	
System-Takt	4 MHz
Arbeitstakt	max. 2 MHz

Strom- und Spannungsversorgung +5 Volt ±5%

$I_{max.}$	1000 mA	ohne Programmspeicher, mit 2 PIAs und 2 K RAM
$I_{typ.}$	500 mA	
$I_{max.}$	750 mA	ohne Programmspeicher, mit 1 PIA und 1 K RAM
$I_{typ.}$	400 mA	

Bei Verwendung von EPROMs sind noch zusätzlich folgende Versorgungsspannungen erforderlich:

+ 12 Volt ± 5%

I_{max}	180 mA*
$I_{typ.}$	105 mA*

– 5 Volt +5%

$I_{max.}$	68 mA*
I_{typ}	35 mA*

*)Programmspeicher mit 4 x TMS 2716 JL

Arbeitstemperaturbereich . . . . 0 bis 50 °C



Telefon 089/61 18-1
Telex 5-22106
Eschenstraße 2
8021 Taufkirchen/München

Verlagsprogramm

Übersicht lieferbarer Bücher

Bestell Nr.	ISBN	Verfasser	Titel	Preis DM
1	3-921682-01-0	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 1	19,80
2	3-921682-02-9	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 2	19,80
3	3-921682-03-7	Gebauer	Elektronik im Auto	9,80
4	3-921682-04-5	Lorenz	IC-Handbuch, TTL, CMOS, Linear	19,80
5	3-921682-05-3	Steinbach	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	3-921682-06-1	Steinbach	IC-Schaltungen, TTL, CMOS, Linear	9,80
7	3-921682-33-9	Hofacker	Elektronik Schaltungen	5,–
8	3-921682-08-8	Lorenz	IC-Bauanleitungs-Handbuch	19,80
9	3-921682-09-6	Lorenz	Feldeffekttransistoren	5,–
10	3-921682-34-7	Lorenz	Elektronik und Radio, 4. Auflage	19,80
11	3-921682-11-7	Lorenz	IC-NF Verstärker	9,80
12	3-921682-12-6	Bernstein	Beispiele Integrierter Schaltungen (BIS)	19,80
13	3-921682-13-4	Lorenz	HEH, Hobby Elektronik Handbuch	9,80
14	3-921682-14-2	Lorenz	IC-Vergleichsliste	29,80
15	3-921682-15-0	Lorenz	Optoelektronik Handbuch	19,80
16	3-921682-16-9	Bernstein	CMOS Teil 1 Einführung Entwurf, Schaltbeispiele	19,80
17	3-921682-17-7	Bernstein	CMOS Teil 2 Entwurf und Schaltbeispiele	19,80
18	3-921682-18-5	Bernstein	CMOS Teil 3 Entwurf und Schaltbeispiele	19,80
19	3.921682-19-3	Lorenz	IC-Experimentier Handbuch	19,80
20	3-921682-20-7	Lorenz	Operationsverstärker	19,80
21	3-921682-21-5	Lorenz	Digitaltechnik Grundkurs	19,80
22	3-921682-22-3	Bernstein	Mikroprozessoren, Eigenschaften und Aufbau 2. Aufl.	19,80
23	3-921682-23-1	Lorenz	Elektronik Grundkrus Kurzlehrgang Elektronik	9,80
24	3-921682-35-5	Hans Peter Blomeyer-Bartenstein	Mikrocomputer Technik	29,80
25	3-921682-25-8	C. Lorenz	Hobby Computer Handbuch	29,80
26	3-921682-26-8	H. Bernstein	Mikroprozessor Teil 2	19,80
27	3-921682-27-4	C. Lorenz	Mikrocomputer Software Handbuch	29,80
28	3-921682-28-2	C. Lorenz	Lexikon + Wörterbuch für Elektronik und Mikroprozessortechnik LEM	29,80
29	3-921682-29-0	C. Lorenz	Mikrocomputer Datenbuch	49,80
30	3-921682-30-4	C. Lorenz	Aktivtraining-Mikrocomputer (Ordner)	49,80
31	3-921682-31-2	C. Lorenz	57 Programme in BASIC (Cames)	39,–
32	3-921682-32-0	C. Lorenz	Circuits Digital Et Pratique	19,80
33	3-921682-36-6	Dr. Hatzenbichler	Microcomputer Programmierbeispiele	19,80
41	–	–	Experimentierplatine für 14, 16, 24, 28 und 40 polige DIL IC's	79,–
105/1	–	–	TTL-Experimentierbuch	5,–
106/1	–	–	CMOS-Experimentierbuch	5,–
109	3-921682-43-6	P. Heuer	6502 Microcomputer Aufbau und Programmierung	29,80
110	3-921682-49-5	C. Lorenz	Programmierhandbuch für PET	29,80
113	3-921682-48-7	C. Lorenz	BASIC Programmierhandbuch	19,80

Ing. W. Hofacker GmbH Verlag
8 München 75